PERFORMANCE SCALABILITY OF N-TIER APPLICATION IN VIRTUALIZED CLOUD ENVIRONMENTS: TWO CASE STUDIES IN VERTICAL AND HORIZONTAL SCALING

A Thesis
Presented to
The Academic Faculty

by

Junhee Park

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Computer Science

Georgia Institute of Technology
May 2016

PERFORMANCE SCALABILITY OF N-TIER APPLICATION IN VIRTUALIZED CLOUD ENVIRONMENTS: TWO CASE STUDIES IN VERTICAL AND HORIZONTAL SCALING

Approved by:

Professor Dr. Calton Pu, Advisor
School of Computer Science
Georgia Institute of Technology

Professor Dr. Ling Liu
School of Computer Science
Georgia Institute of Technology

Professor Dr. Qingyang Wang
School of Electrical Engineering and
Computer Science
Louisiana State University

Professor Dr. Shamkant B. Navathe
School of Computer Science
Georgia Institute of Technology

Professor Dr. Edward R. Omiecinski
School of Computer Science
Georgia Institute of Technology

Date Approved: December 11, 2015

To my parents, my wife, and my daughter

ACKNOWLEDGEMENTS

My Ph.D. journey was a precious roller-coaster ride that uniquely accelerated my personal growth. I am extremely grateful to anyone who has supported and walked down this path with me. I want to apologize in advance in case I miss anyone.

First and foremost, I am extremely thankful and lucky to work with my advisor, Dr. Calton Pu who guided me throughout my Masters and Ph.D. programs. He first gave me the opportunity to start work in research environment when I was a fresh Master student and gave me an admission to one of the best computer science Ph.D. programs in the world. I could not imagine where I am now when I first came to his office door step. It was his constant guidance and encouragement that have been the most important driving force for my growth. He quickly grasped my thinking and guided me toward shaping concrete idea out of it no matter how vague and unclear it was. Not only professional advices, Calton did not hesitate to give life wisdom based on his experience. I have learned a countless amount and I am indebted to him for his support throughout my years at Georgia Tech.

I am also grateful to Dr. Qingyang Wang, my long-term research mentor, colleague and friend who is now a professor in Louisiana State University. Qingyang and I came to Georgia Tech at the same year 2007. He was a PhD. student and I was master student. We soon became very good friends as we share common interest not to mention his good personality. Since then he has been the mentor who lead the way toward the ultimate goal as well as a friend who accompanied me through the peaks and valleys of my Ph.D. journey. I'm very fortunate to have him throughout my years in Georgia Tech and grateful for all his guidance and everything I have learned from him.

I would also like to give special thanks to the members of my dissertation committee - Dr. Ling Liu, Dr. Shamkant Navathe, Dr. Edward Omiecinski, and Dr. Qingyang Wang - for serving on my dissertation committee. I appreciate all the challenging questions before and during my defense that led to an improved dissertation and approving this work.

To my previous and current colleagues in ELBA project Jack Li, Chien-An Lai, Tao Zhu, Chien-An Cho, Simon Malkowski, Deepal Jayasinghe, Pengcheng Xiong, Aibek Musaev, Gueyoung Jung, Qinyi Wu, and Younggyun Koh you have supported me all along the way. We all have been in this Ph.D. boat together and without your company, I could not have gone through all the hardships. I give special thanks to Jack for his suggestions and patience with all my baby mama dramas as well as my research. His suggestions made me resolve issues in the better way and overcome the difficulties.

Also during my graduate studies in Georgia tech, I was very fortunate to meet many awesome friends who have peppered my Ph.D. journey with cherished memories, including Yasuhiko Kanemasa, Chiemi Amagasa, Chris Grayson, Binh Han, Danesh Irani, Minsung Jang, Mukil Kesavan, Balaji Palanisamy, Priyanka Tembey, De Wang, Jinpeng Wei, and Rui Zhang. My deepest thanks to you guys.

Lastly, but most significantly, to my parents and my family, I was able to study here and pursue my goal because of your unconditional support. I love you all very much. To my mother, Mija Ahn, you are my strongest supporter no matter what situation I am in. To my father, Seok Cheon Park, I thank you for being my lighthouse whenever I felt I am lost. To my sister Kyung-Jin Kim, I appreciate your encouragements throughout my PhD study. To my dearest wife Hee Rin Lee, thank you for accompanying with me and supporting me through my Ph.D. journey. I sincerely respect how much you have sacrificed for both me and our daughter. Without all the support you have given me, I would not have made it this far. My daughter Claire Gaeun Park, I am sorry daddy was not always there for you and I love you with all my heart.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

The prevalence of multi-core processors with recent advancement in virtualization technologies has enabled horizontal and vertical scaling within a physical node achieving economical sharing of computing infrastructures as computing clouds. Through hardware virtualization, consolidated servers each with specific number of core allotment run on the same physical node in dedicated Virtual Machines (VMs) to increase overall node utilization which increases profit by reducing operational costs. Unfortunately, despite the conceptual simplicity of vertical and horizontal scaling in virtualized cloud environments, leveraging the full potential of this technology has presented significant scalability challenges in practice. One of the fundamental problems is the performance unpredictability in virtualized cloud environments (ranked fifth in the top 10 obstacles for growth of cloud computing). In this dissertation, we present two case studies in vertical and horizontal scaling to this challenging problem. For the first case study, we describe concrete experimental evidence that shows important source of performance variations: mapping of virtual CPU to physical cores. We then conduct an experimental comparative study of three major hypervisors (i.e., VMware, KVM, Xen) with regard to their support of n-tier applications running on multi-core processor. For the second case study, we present empirical study that shows memory thrashing caused by interference among consolidated VMs is a significant source of performance interference that hampers horizontal scalability of an n-tier application performance. We then execute transient event analyses of fine-grained experiment data that link very short bottlenecks with memory thrashing to the very long response time (VLRT) requests. Furthermore we provide three practical techniques such as VM migration, memory reallocation, soft resource allocation and show that they can mitigate the effects of performance interference among consolidate VMs.

CHAPTER I

INTRODUCTION

The prevalence of multi-core processors (MCPs) with recent advancement in virtualization technologies has enabled horizontal and vertical scaling within a physical node achieving economical sharing of computing infrastructures as computing clouds. Through hardware virtualization, consolidated servers each with specific number of core allotment run on the same physical node in dedicated Virtual Machines (VMs) to increase overall node utilization which increases profit by reducing operational costs [38]. Unfortunately, despite the conceptual simplicity of vertical and horizontal scaling in virtualized cloud environments, leveraging the full potential of this technology has presented significant scalability challenges in practice. In fact, enterprise computing infrastructures continue to struggle with surprisingly low resource utilization [31, 84]. One of the fundamental problem is the performance unpredictability in virtualized cloud environments (ranked fifth in the top 10 obstacles for growth of cloud computing [14]).

There are numerous factors that may contribute to the apparent unpredictability of n-tier application performance (e.g., response time, throughput) when scaling in virtualized cloud environments. These factors, which we call Quality of Services (QoS) determinants, include but are not limited to hardware and software components (e.g., multi-core processor, hypervisor, server thread pool size), system state (e.g., data size and distribution),and workload (e.g., workload transaction mix). For example, CPU cache miss (e.g., Last Level Cache miss) or application level cache miss (e.g., database cache or web server cache) create significant variance in performance measurement making it difficult to provide predictable application performance [54, 69, 76]; Performance interference among consolidated applications have been demonstrated variety of concrete systems and applications [11, 42, 51, 73] adding more difficulties. Due to the complex dependencies among the hardware and software components in the system [93], the unpredictability of n-tier application performance

when scaling in virtualized cloud environments is a compound effect of all these factors.

In my dissertation research, we present two case studies in vertical and horizontal scaling to this challenging problem. For the first case study, we describe concrete experimental evidence that shows important source of performance variations: the mapping of physical cores to virtual [1] CPUs (vCPU) can significantly lower on-chip cache hit ratio, causing performance drops of up to 22% without obvious changes in resource utilization. After we eliminated these variations by fixing the MCP core mapping, we measured the impact of three mainstream hypervisors (the dominant Commercial Hypervisor [2] (CH), Xen [2], and KVM [1]) with regard to their support of n-tier applications running on multi-core processors. For the second case study, we present empirical study that shows memory thrashing caused by interference among consolidated VMs is a significant source of performance interference that hampers horizontal scalability of an n-tier application performance. Specifically, we describe non-monotonic performance scalability and detailed analyses on its root cause: large number of page faults (i.e., memory thrashing) caused by interference among consolidated VMs induces IO queue overflows in hypervisor's storage stack resulting in very long response time requests due to frequent CPU IOwait. We then conduct transient event analyses of fine-grained experiment data that link very short bottlenecks with memory thrashing to the very long response time (VLRT) requests. Concretely, we present frequent CPU IOwait caused by memory thrashing in database tier induces request queue overflows that propagate through the entire n-tier system, resulting in very long response time requests due to frequent TCP re-transmissions. Furthermore we provide three practical techniques for the memory thrashing problem induced by VM interference, including VM migration, memory re-allocation, soft resource re-allocation and show that they can mitigate the effects of performance interference among consolidate VMs.

As we combine fine-grained monitoring tools (a combination of microsecond resolution message time-stamping and millisecond system resource sampling) and empirical analysis

[1] We use the term *virtual* to refer to hardware components of a virtual machine (VM).

[2] Due to licensing and copyrights issues which prevent publications of performance or comparison data, we mask our choice of commercial virtualization technology. We use commercial hypervisor or CH interchangeably throughout the proposal.

methods to generate and analyze monitoring data, we are able to study, understand, and address scalability challenges in a systematic way.

1.1 Dissertation Statement and Contributions

My dissertation statement is formulated as follows:

Thesis Statement: *The performance scalability of elastic n-tier application on modern virtualized compute cloud presents scalability challenges that can be studied, understood, and addressed through an empirical approach based on fine-grained analysis of the system.*

To support my dissertation statement, we make the following concrete contributions:

- **Our first contribution is an analysis of intra-MCP scalability (vertical scalability) of n-tier application in virtualized cloud environments (Chapter 2).** As a first step, we show the variation in n-tier application performance induced by core mapping. One of the key outcome is the identification of an important source of measurement variance: mapping of vCPUs to physical cores. For a dual-CPU MCP, mapping four vCPUs to a single CPU (with four cores on the same memory bank and cache) produced significant performance gains compared to mapping four vCPUs to two CPUs that do not share cache. The performance gains of up to 22% are achieved without any obvious sign of resource under-utilization, since cores remain "busy" when cache misses and they have to await the memory access. By pinning vCPUs to appropriate physical cores we were able to eliminate this source of variance and accurately compare the performance differences due to three mainstream hypervisors (Section 2.3).

  As a second step, we show an experimental comparative study of three major hypervisors with regard to their support of n-tier applications running on multi-core processors. Concretely, we ran an n-tier application benchmark (RUBBoS [3]) on three major hypervisors (i.e., the dominant Commercial Hypervisor (CH), XEN, and KVM) to compare their support of intra-MCP performance scalability. Our data illustrate that the hypervisors showed both similarities and dissimilarities.

For similarities, we observed a non-monotonic scalability trend to multi-cores across all three hypervisors when running a browse-only CPU-intensive workload. For instance, we found that the system was only able to scale linearly up to four cores and five cores onward the throughput starts to deteriorate while all cores were still fully utilized. This problem can be traced to the CPU overhead induced by inefficient management of the last level cache (LLC) in CPU packages (Section 2.4). For dissimilarities, we found that each hypervisor has its own strategy of handling write operations (i.e., I/O). Due to this difference, we observed significant performance differences among hypervisors when running a mixed read/write I/O-intensive workload. Specifically, XEN outperforms the Commercial Hypervisor by 22% and the Commercial Hypervisor is able to provide more than twice the throughput compared to KVM (Section 2.5).

- **Our second contribution is the empirical study of the performance interference among consolidated VMs and its impact on horizontal scalability of n-tier application performance (Chapter 3).** One of the key outcome of this work is the identification of an important source of performance interference: memory thrashing caused by interference among consolidated VMs. We start by showing a non-monotonic performance scalability when we scale the system horizontally by adding more server VMs within a physical host. For a physical host with 16GB ram, four consolidated VMs each with 2GB memory experienced significant performance loss compared to three consolidated VMs. The performance loss of up to 50% are achieved without any obvious resource bottlenecks. Timeline analysis showed two distinct operational modes within a typical RUBBoS benchmark: the first half of the runtime session showed frequent CPU IOwait causing very long response time (VLRT) requests although the entire dataset is in memory and the workload is readonly. Surprisingly such abnormal CPU IOwait disappeared in the second half resulting moderate utilization when averaged. Subsequent analysis of hypervisor's IO mechanism showed large number of page faults (i.e., memory thrashing) caused by interference among consolidated VMs induces IO queue overflows in hypervisor's storage stack resulting in very long response time requests due to frequent CPU IOwait although we

explicitly mapped virtual to physical resources to remove performance variation [69] and memory was not over-committed. (Section 3.3).

We then present the three practical techniques that can reduce the severity of memory thrashing interference or mitigate its effect on the whole system performance. For example, we show that 1) VM migration and 2) memory re-allocation are simple and work well when more computing power is available. We also show that by reducing the concurrency of the system through 3) soft resource re-allocation [94], we can remedy the interference when more resources are unavailable (Section 3.4).

- **Our thrid contribution is a set of transient event analyses of fine-grained experimental data that link very short bottlenecks with memory thrashing to the very long response time (VLRT) requests (Chapter 4).** The key outcomes of this work are identification of the root cause of the very short bottleneck (i.e., memory thrashing) and associating it with poor n-tier application performance (i.e., VLRT requests) thus confirming our findings in Chapter 3. The steps of our transient event analyses follow closely to the earlier study conducted by Wang et al. [90]. First we observe very long response time (VLRT) requests while the system is under moderate utilization. Second, using ElbaLens - a light-weight request tracing tool, we discover long request queues are formed in the Apache overflowing TCP buffer, causing re-transmission during that time. The long queues in Apache are formed because of the queue overflows in saturated downstream tier (i.e., DB tier in our work) that propagate through the entire n-tier system. Third, the long queues in DB servers are created by very short CPU IOwait bottlenecks, in which the server CPU becomes saturated for a very short period of time due to frequent CPU IOwait that is CPU waiting IO operation to complete. Fourth, through extensive measurements of an n-tier application benchmark (RUBBoS [3]), we identified that memory thrashing is the root cause associated with the very short bottleneck (Section 4.3).

1.2 Organization of This Dissertation

The rest of this dissertation is organized as follows. Chapter 2 presents the intra-MCP (Multi-Core Processor) scalability of n-tier application in virtualized cloud environments. Chapter 3 illustrates empirical study of the impact of memory thrashing caused by interference among consolidated VMs when we scale horizontally in virtualized cloud environments. Chapter 4 describes the transient event analyses that explicitly link memory thrashing to very long response time (VLRT) requests, based on the fine-grained measurement data collected from different system layers. Chapter 5 summarizes the related work and Chapter 6 concludes the dissertation and discusses future work.

CHAPTER II

VARIATIONS IN PERFORMANCE MEASUREMENTS OF MULTI-CORE PROCESSORS

This chapter presents first case study of my core thesis research that is intra-MCP (Multi-Core Processor) scalability of n-tier application in virtualized cloud environment. First, Section 2.3 illustrates the performance variance caused by hypervisor's mapping virtual CPUs (vCPUs) of a virtual machine (VM) into physical cores. Second, Section 2.4 and Section 2.5 presents experimental comparative study of three major hypervisors with regard to their support of n-tier applications running on multi-core processor. (Section 2.4 on non-monotonic system scalability, and Section 2.5 on performance impact of underlying virtualization technologies).

2.1 Introduction

Providing predictable quality of service (QoS) is an important goal for web-facing applications in cloud environments. The prevalence of multi-core processors (MCPs) in computing clouds and data centers today has raised the question of whether applications can use the increasing number of cores efficiently in order to provide predictable QoS. On a physical chip, an MCP often has a hierarchical organization with multiple CPUs each having multiple cores. A dual-CPU quad-core processor would have 2 CPUs, each with four cores. One of the most important challenges in clouds and data centers is the intra-MCP horizontal scalability of system software. This is often attempted through virtualization, where a hypervisor maps virtual CPUs (vCPU) of a virtual machine (VM) into physical cores. The intra-MCP horizontal scalability is a particular problem for n-tier applications and systems due to the several layers and various system component choices. In this paper, we run extensive experiments to measure and compare the intra-MCP horizontal scalability of n-tier server components.

The first contribution of the chapter is the identification of an important source of measurement variance: mapping of vCPUs to physical cores. For a dual-CPU MCP, mapping 4 vCPUs to a single CPU (with 4 cores on the same memory bank and cache) produced significant performance gains compared to mapping 4 vCPUs to two CPUs that do not share cache. The performance gains of up to 22% are achieved without any obvious sign of resource under-utilization, since cores remain "busy" when cache misses and they have to await the memory access. By pinning vCPUs to appropriate physical cores we were able to eliminate this source of variance and accurately compare the performance differences due to three mainstream hypervisors (Section 2.3).

The second and main contribution of the chapter is an experimental comparative study of 3 major hypervisors with regard to their support of n-tier applications running on MCP. Concretely, we ran an n-tier application benchmark (RUBBoS [3]) on 3 major hypervisors (i.e., the dominant Commercial Hypervisor, XEN, and KVM) to compare their support of intra-MCP horizontal performance scalability. Our data shows that the hypervisors showed both similarities and dissimilarities.

For similarities, we observed a non-monotonic scalability trend to multi-cores across all three hypervisors when running a browse-only CPU-intensive workload. For instance, we found that the system was only able to scale linearly up to four cores and five cores onward the throughput starts to deteriorate while all cores were still fully utilized. This problem can be traced to the CPU overhead induced by inefficient management of the last level cache (LLC) in CPU packages (Section 2.4). For dissimilarities, we found that each hypervisor has its own strategy of handling write operations (i.e., I/O). Due to this difference, we observed significant performance differences among hypervisors when running a mixed read/write I/O-intensive workload. Specifically, XEN outperforms the Commercial Hypervisor by 22% and the Commercial Hypervisor is able to provide more than twice the throughput compared to KVM (Section 2.5).

Our empirical analysis suggests that in order for enterprise n-tier applications to better scale in multi-core virtualization environments, both MCP cache architecture and the choice of hypervisors should be considered integral components. However despite their differences,

performance scalability of n-tier applications beyond four cores remains a challenge for all three mainstream virtualization technologies.

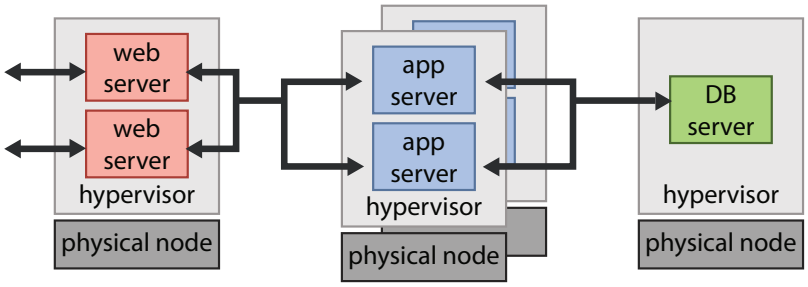
The remainder of this chapter is organized as follows. The next section presents the description of our experimental setup including the profiling environments, tools and benchmark. Subsequently, Section 2.3 introduces the performance variation induced by core mapping. Section 2.4 describes a non-monotonic scalability trend to multi-core processors. Section 2.5 illustrates performance difference among hypervisors. Finally, we present the related work in Section 2.6 and conclude our work in Section 2.7.

2.2 Experimental Setup

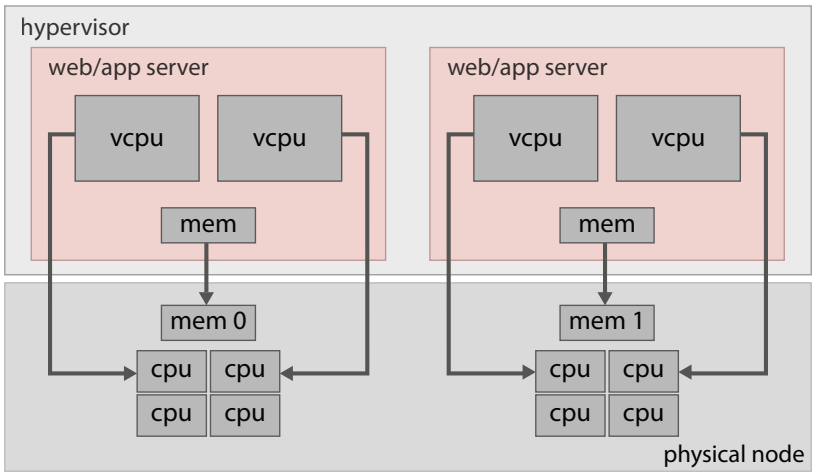
While the intra-MCP horizontal scalability may be evaluated using any type of application, the focus of this paper is n-tier applications with LAMP (Linux, Apache, MySQL, and PHP) implementations. Typically, n-tier applications are organized as a pipeline of servers[1], starting from web servers (e.g., Apache), through application servers (e.g., Tomcat), and ending in database servers (e.g., MySQL). This organization, commonly referred to as n-tier architecture (e.g., 3-tier in Figure 1(a)), serves many important web-facing applications such as e-commerce, customer relationship management, and logistics.

In our experiments, we adopt the RUBBoS n-tier benchmark, based on bulletin board applications such as Slashdot [3]. RUBBoS has been widely used in numerous research efforts due to its real production system significance. The workload includes 24 different interactions such as "register user" or "view story". The benchmark includes two kinds of workload modes: browse-only and read/write interaction mixes. Our default experiment trial consists of a three-minute ramp-up, a three-minute runtime, and a 30-second ramp-down. We run the RUBBoS benchmark in a 3-tier system (Figure 1(a)) with workload raging from 1000 up to 35000 users while scaling the system from one up to eight cores under four different environments. We exploit hardware-assisted VM (HVM) and performance measurements (e.g., CPU utilization) are taken during the runtime period using Sysstat and Collectl at one and 0.1 second granularity respectively. We utilize OProfile and Xenoprof

[1]In this paper, server is used in the sense of computer programs serving client requests. Hardware is referred to as a physical computing node or node for short.



(a) 3-tier application system with seven servers (i.e., web, application, and database) and four physical hardware nodes in total. Two dedicated web/app server VMs are co-located on a single physical hardware node.



(b) Details of resource mapping between web/app server VMs and a shared physical hardware node. The VMs' virtual CPUs and memory are explicitly mapped to separate physical CPU packages and memory banks to mitigate interference.

Figure 1: Example of a 3-tier application system deployment, presented as mappings of virtual machines to physical hardware nodes.

to monitor last level cache misses from host.

At an abstract level, the deployment of n-tier applications in a cloud computing infrastructure can be modeled as a mapping between component servers and physical computing nodes. Figure 1(a) exemplifies our n-tier application deployment with two web server virtual machines (VMs), four application server virtual machines, and one dedicated database server virtual machine. Other than the database server VM, each server VM has two virtual cores (virtual CPUs), 2GB of memory, and 20GB HDD and we co-located two server VMs on a single physical node using the dominant Commercial Hypervisor[2] (CH) as illustrated in

[2]Due to licensing and copy rights issues which prevent publications of performance or comparison data,

Table 1: Summary of experimental setup (i.e., hardware, operating system, software, and virtualization environments).

CPU	Quad Xeon 2.27GHz * 2 CPU (8M L3 Cache)
Memory	16GB (8GB per Memory Bank)
HDD	SATA, 7200RPM, 500GB
Network I/F	1Gbps
Web Server	HTTPD-2.2.22
App Server	Apache Tomcat-5.5.17
Connector	Tomcat Connectors-1.2.32-src
DB Server	Mysql-5.5.28-linux2.6-x86_64
Java	JDK-1.6_23
Monitoring Tools	Sysstat, OProfile(Xenoprof), Collectl
Hypervisor	Commercial Hypervisor (CH), KVM, XEN
Virtualization Type	Full virtualization (HVM)
Guest OS	RHEL Server 6.3 64-bit
Guest OS Kernel	2.6.32-279.19.1.el6.x86_64

the Figure 1. In order to mitigate interference between the two co-located VMs, each VM's virtual resources are explicitly mapped (i.e., pin) to separate physical CPU packages and memory banks as shown in Figure 1(b). Through extensive experiments we confirmed that this topology does not introduce any artificial bottlenecks induced by VM co-location in web-tier and application-tier; however these empirical results are omitted here due to space constraints. Other important characteristics of our experimental testbed are summarized in Table 1.

2.3 Performance Variations Induced by Core Mapping

When we started running the experiments outlined in Sections 2.2 and 2.4, we encountered significant variance in the measurement results as we repeated exactly the same experiments (hardware and software). This variance (up to more than 20%) has been reported anecdotally in various cloud environments for a variety of benchmarks, but to the best of our knowledge its sources have yet to be unambiguously identified. This variance is a significant problem in the analysis of our experimental results, since it is about the same order

we mask our choice of commercial virtualization technology. We use commercial hypervisor or CH interchangeably throughout the paper.

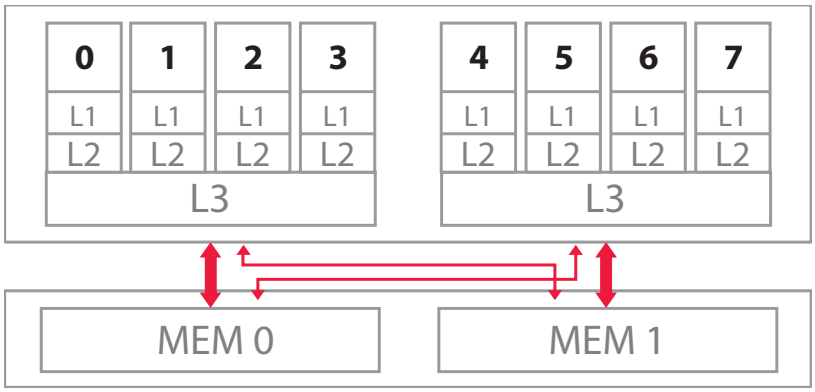
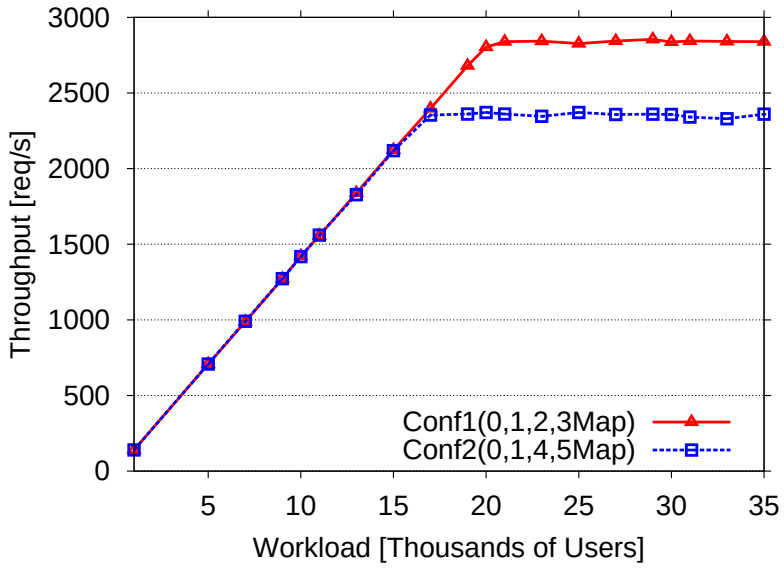


Figure 2: Schematic illustration of CPU and memory architecture in our experimental testbed.

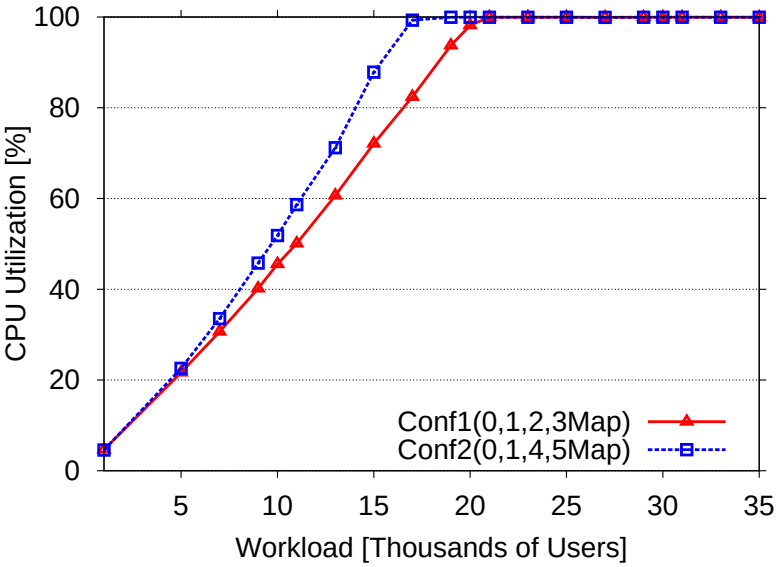
of magnitude as the measured differences among the hypervisors.

Through the data collection from a large number of experiments and careful analysis, we found that the cache hit ratio that influences overall application performance (described in Section 2.4) also seem to be connected to the variances in the repeated measurements. Since the repeated experiments run on exactly the same hardware and software configurations, we concluded that the "hidden variable" must reside at a level below the typical hardware configuration settings. A careful study of the CPU cache architecture (outlined schematically in Figure 1(b) and Figure 2) showed that the mapping of virtual CPUs (vCPU) in VMs to physical cores is not necessarily static for every hypervisor, unless they are explicitly "pinned". By comparing the measured performance of experiments with different mappings of vCPUs to physical cores, we were able to find the main source of the variances observed: the cache hit ratio.

The mapping/pinning of vCPUs to physical cores has significant influence on cache hit ratio because the two physical CPUs have separate memory banks and caches. Therefore, only if all vCPUs (up to four) are mapped to the four cores of a single CPU will they share the same L3 cache (Figure 2). Using a concrete scenario to illustrate the problem, let us consider two different mapping strategies for 4 cores under the XEN environment. The first configuration maps all four vCPUs into a single CPU package, which are cores numbered $0, 1, 2, 3$ for Xen. The second configuration maps two vCPUs into each one of the two CPU packages (e.g., cores $0, 1$ in one CPU and cores $4, 5$ in the other CPU).



(a) Throughput comparison between the two 4-core mappings (4 cores in 1 CPU package vs 2 cores in 2 CPU packages); VMs mapped with cores in same CPU package provides 22% more performance than VMs with cores mapped to different CPU packages.



(b) CPU utilization comparison between the two 4-core mappings (4 cores in 1 CPU package vs 2 cores in 2 CPU packages); While both mappings report 100% CPU utilization, cores mapped to different CPU packages saturate earlier.

Figure 3: Analysis of performance variation induced by core mapping.

Figure 3(a) illustrates throughput comparison between the two mapping cases when scaling from 1000 to 35000 users. We found that the single-CPU mapping outperforms the

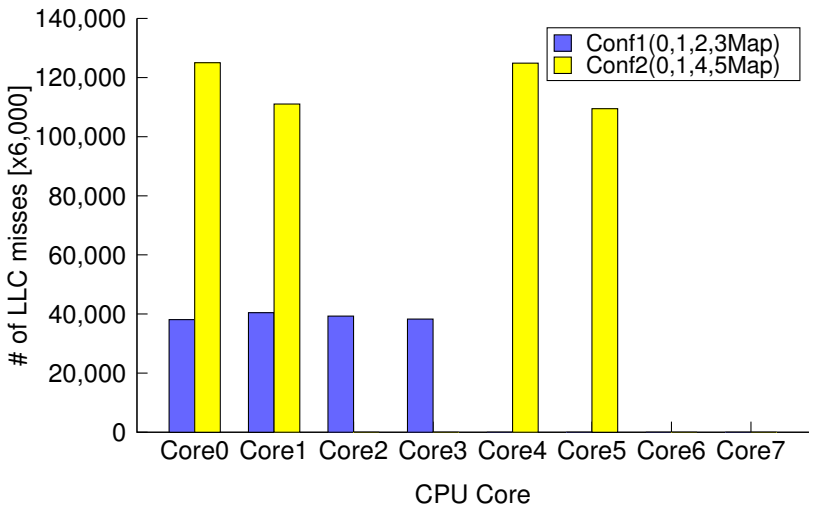
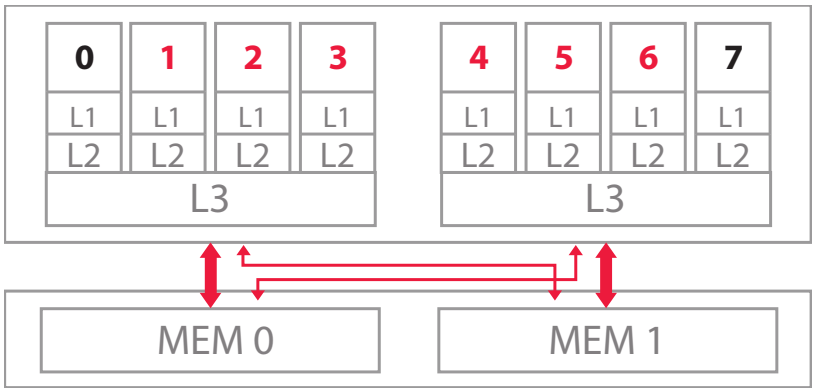


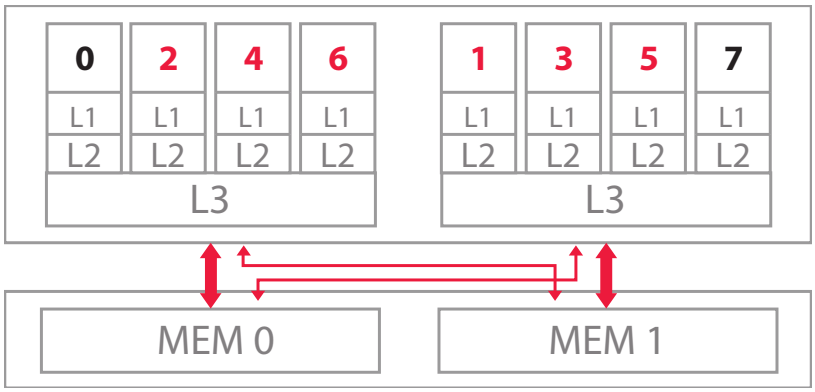
Figure 4: Comparison of LLC misses per core between the two 4-core mappings; Cores in two different CPU packages showed overall 128% more LLC misses and the cores used in both mappings get less efficient due to significant increase in LLC misses.

two-CPU mapping by 22%. Interestingly, the performance gap between the two configurations happened without any obvious under-utilization of system resources that may have contributed to this performance penalty. Figure 3(b) shows the CPU utilization comparison between the two mapping strategies. Both mappings show all cores being fully utilized, but cores in the two-CPU mapping saturate much earlier than those from the single-CPU mapping. Further analysis of measurement data revealed that the two-CPU mapping cores have 128% higher LLC misses than the single-CPU mapping. By identifying the LLC misses of individual cores, we also observed that cores that were used in both cases such as Core0 and Core1 show significant increase in LLC misses (i.e., 228%, 175% respectively) in the two-CPU mapping (Figure 4). These results suggest strongly that the mapping of vCPUs to physical cores can have significant impact on the n-tier application performance due to cache management issues.

From an operational point of view, the mapping through pinning of vCPUs to physical cores is a non-trivial task. First, the physical pinning facility may not be available to a normal user of computing clouds. Often it requires administrator privileges. Second, different hypervisors have different core number assignment strategies as shown in Figure 5. Concretely, CH and XEN assign core numbers $0, 1, 2, 3$ to CPU0 and core numbers $4, 5, 6, 7$



(a) Core number assignment strategy adopted by CH and XEN.



(b) Core number assignment strategy adopted by KVM and native RHEL and Debian.

Figure 5: Analysis of core number assignment strategy among various platforms.

to CPU1 as shown in Figure 5(a). In contrast, KVM and the native deployment of RHEL6 and Debian assign core numbers $0, 2, 4, 6$ to CPU0 and core numbers $1, 3, 5, 7$ to CPU1 as shown in Figure 5(b). This lack of standard mapping/naming convention required careful mapping specifications at the physical core level to achieve the best cache hit ratios. For example, pinning 4 vCPUs to cores $0, 1, 2, 3$ is a single-CPU mapping for Xen, but a two-CPU mapping for KVM. Conversely, pinning 4 vCPUs to cores $0, 2, 4, 6$ is a single-CPU mapping for KVM, but a two-CPU mapping for Xen.

2.4 Non-Monotonic System Scalability to Multi-Core Environments

In the previous section, we showed that the mapping of vCPUs to physical cores inside a single CPU package produced significant performance gains due to an increase in on-chip cache hit ratio. In this section, we investigate intra-MCP horizontal scalability on four

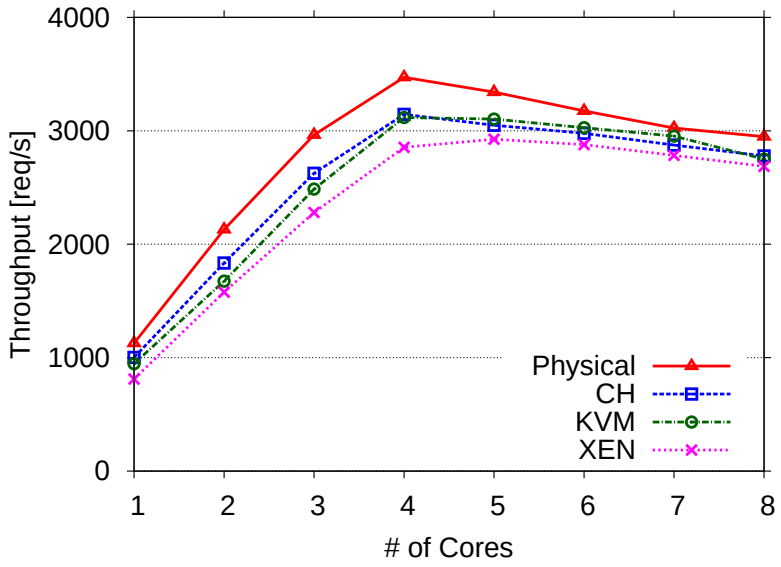
different environments using a browse-only CPU-intensive workload. Through extensive measurements, we found that the system only scales linearly up to four cores, and the performance starts to decrease from five cores onward. We observed this non-monotonic scalability trend appeared on all environments. Section 2.4.1 describes the experimental observation on non-monotonic scalability trend of the system under three mainstream hypervisors (i.e., Commercial Hypervisor, KVM, XEN) as well as the native physical environment. Section 2.4.2 illustrates the impact of the CPU overhead caused by last level cache misses on n-tier application performance and system scalability to multi-core.

2.4.1 Non-Monotonic Trend in Intra-MCP Horizontal Scalability

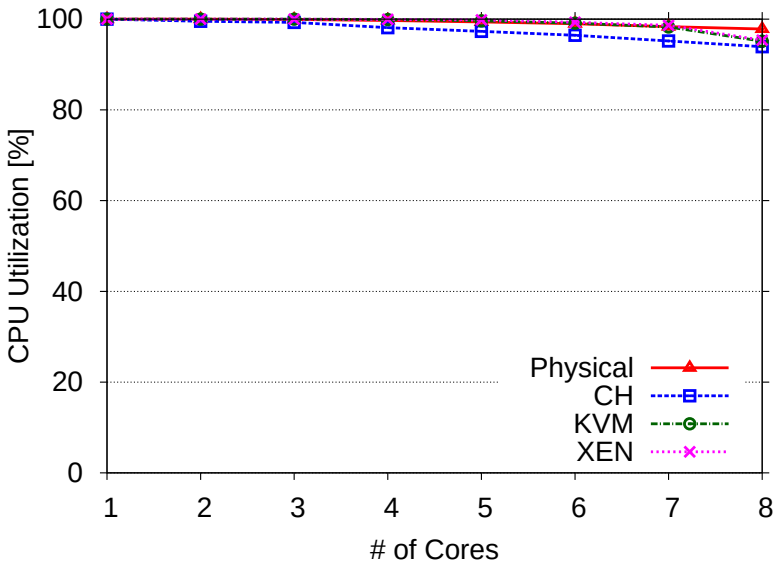
One of the straightforward approaches of scaling an n-tier application in modern cloud platforms is to scale up (e.g., adding more CPUs or memory). While the growth of virtualization technologies and multi-core processors has facilitated the scale up of existing systems, we found that this approach does not always guarantee system scalability.

Figure 6(a) illustrates the performance trend of the system under CH, KVM, XEN, and physical environment. Each throughput value used in the figure represents the maximum throughput that a system can achieve under a given fixed allocation of cores when we scale from 1000 users up to 35000 users. This result shows that the system was able to scale linearly up to four cores producing approximately 50% more throughput as the number of affiliated cores increases; however, from five cores onward, the system stops scaling further and the performance even starts to deteriorate. We found out that this non-monotonic performance scalability trend exists on all three virtual environments. We confirmed that this trend also exists on the physical environment running RHEL6 on bare metal hardware.

We diagnose the cause of the performance degradation by investigating the CPU utilization of the VM. Figure 6(b) depicts the average CPU utilization for each core assignment scenario. It shows that the CPU is fully saturated for all cases when we scale from one core to eight cores. Based on these results we found that the additional CPU power allocated to the VM is being fully utilized, but the performance actually deteriorates. For example, under the eight-core configuration, the average CPU utilization for all eight physical cores



(a) Max throughput of the system under 4 different environments when scaling from 1 core to 8 cores; For all environments, the system shows a non-monotonic scalability trend.



(b) Average CPU utilization when scaling from 1 core to 8 cores; While performance decays from 5 cores onward, all cores are fully utilized.

Figure 6: Analysis of performance variation induced by core mapping.

measured at the host level of XEN, KVM, CH, and physical environment shows 95%, 95%, 94%, and 98%, respectively but performance between the eight core scenario and four core scenario actually decreases by 6%, 12%, 12%, and 15%. Upon further investigation, we found that LLC misses introduce significant CPU overhead which hampers all of the extra

CPU power gained by allocating additional cores to the VM which we discuss in the next section.

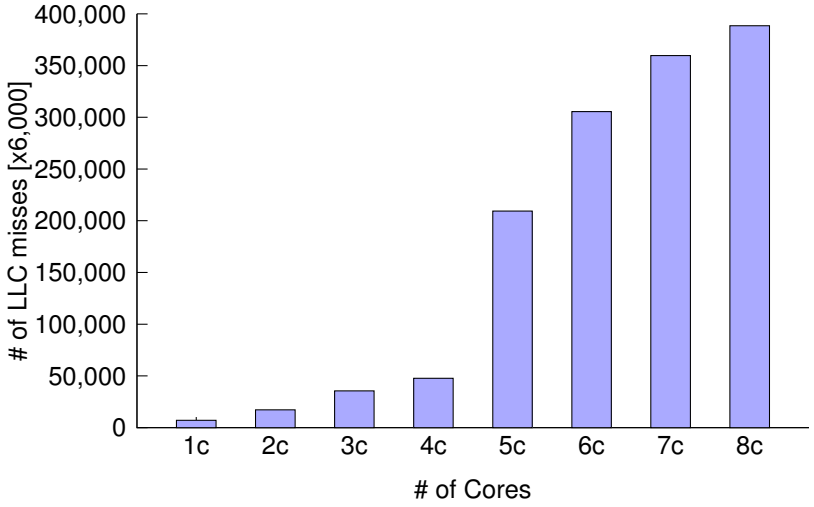
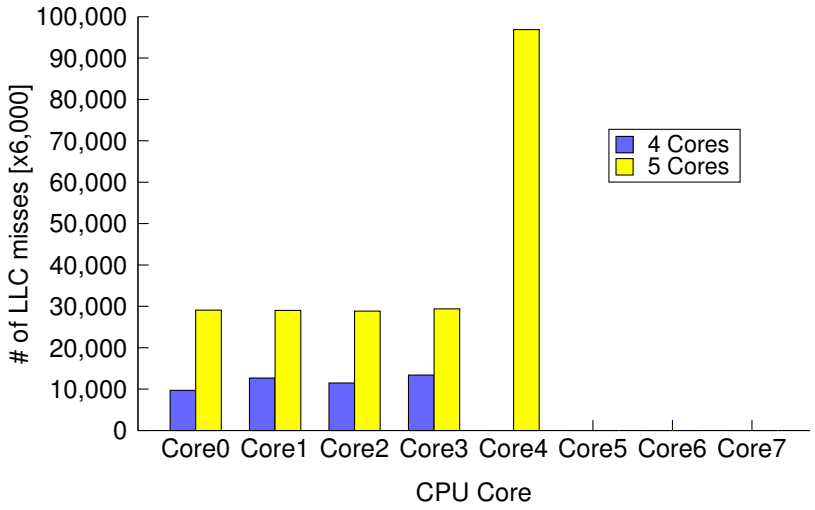


Figure 7: Total number of LLC miss when scaling from 1 core to 8 cores; Exponential increase (390%) is observable after 4 cores.

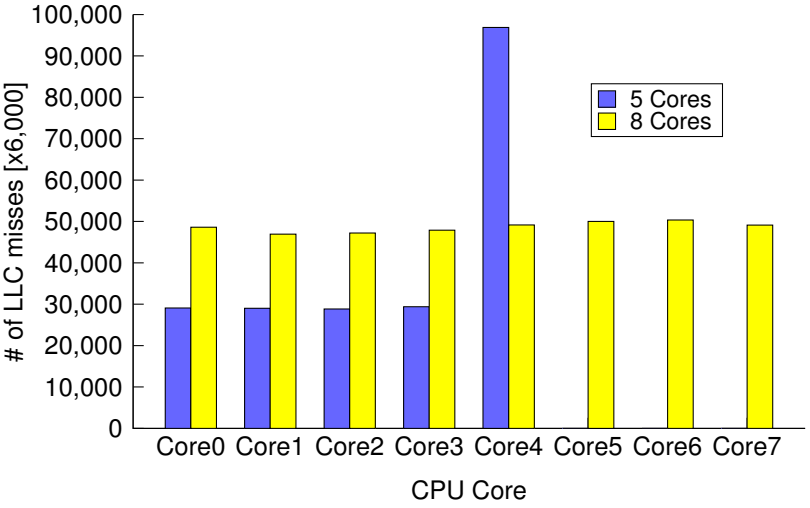
2.4.2 CPU Overhead Induced by Last Level Cache Misses

For modern computer architectures, caching effectiveness is one of the key factors for system performance [27, 54, 91]. In order to monitor the last level cache, we used OProfile and Xenoprof which utilize CPU performance counter[3]. Figure 7 shows the total number of LLC misses (i.e., L3 cache miss) under a given fixed allocation of cores. The number of LLC misses increases linearly from one core to four cores, but we observed a sudden dramatic increase after four cores. For instance, from four cores to five cores the total number of LLC misses increases 390%. Breaking down the total number of LLC misses into individual cores revealed that on-chip cache miss ratio increased significantly for the same cores that were used in the four core allocation scenario as shown in Figure 8(a). For example, Core0 which was used in both cases showed a 200% increase in LLC misses when we scaled from four to five cores. We also observed a notable number of LLC misses on Core4. This is due to Core4 being located in a different CPU package. As explained in Section 2.3, since only Core4 uses separate L3 cache and memory bank, an increase in cache miss ratio is

[3]The CPU performance counter increases by 1 for every 6000 L3 cache misses in our environmental settings.



(a) Comparison of LLC misses per core between 4-core case and 5-core case; Notable number of LLC misses on Core4 due to it being located in a different CPU package and Cores used in both cases are less efficiently utilized.



(b) Comparison of LLC misses per core between 5-core case and 8-core case; Total number of LLC misses increases 86% while significant decrease in LLC misses on Core4 as more cores from 2nd CPU package get allocated.

Figure 8: Analysis of non-monotonic scalability trend of the system through LLC misses.

unavoidable. As more cores from the second CPU package gets allocated, we could observe a significant decrease in LLC misses on Core4 as shown in Figure 8(b).

An experienced reader may immediately question the wisdom of LLC misses when scaling from seven to eight cores, which appears to be an insignificant increase (i.e., 8%) compared to the four to five cores (i.e., 390%). The focus of this section is to show that LLC

misses have significant impact on the performance of n-tier system and its scalability to multi-cores. There are many other factors limiting system scalability such as spin locks and implementation issues residing in the application [23].

Based on these results we realized that while additional CPU cores were allocated and fully utilized, the extra processing power was compensated due to the overhead introduced by LLC misses. Considering the actual number of LLC misses (i.e., LLC miss value times 6000), the high number of the LLC misses cause frequent CPU stalls which wastes its CPU cycles on waiting for the cache line thus canceling the extra CPU power and even degrading the performance.

2.5 Performance Impact of Underlying Virtualization Technologies

So far we have discussed that core mapping can cause significant performance variations due to its influence on cache hit ratio and under browse-only CPU-intensive workload, we observed a non-monotonic scalability trend on all three hypervisors induced by inefficient last level cache (LLC) management. In this section, we focus on the empirical analysis of performance differences among three hypervisors when running a mixed read/write I/O-intensive workload. The results here are based on the same configuration as shown in Section 2.4 except the workload characteristics.

When we scale from one core up to eight cores using I/O intensive workload, we observed two interesting phenomena - limited intra-MCP horizontal scalability and significant performance differences among three hypervisors. Figure 9 depicts the maximum achievable throughput of the system under three different hypervisors (i.e., XEN, CH, KVM) when scaling from one core up to eight cores. This figure clearly shows our two observations. Firstly, the system does not scale well to multiple cores. For instance, under XEN and CH, the system was only able to scale up to three cores and two cores respectively. Moreover, the KVM system showed no throughput gains regardless of how many cores were allocated to it. Secondly, there are significant performance differences among three hypervisors. For example, XEN outperforms the Commercial Hypervisor by 22% and the CH is able to provide more than twice the throughput compared to KVM. Let's look into each observation

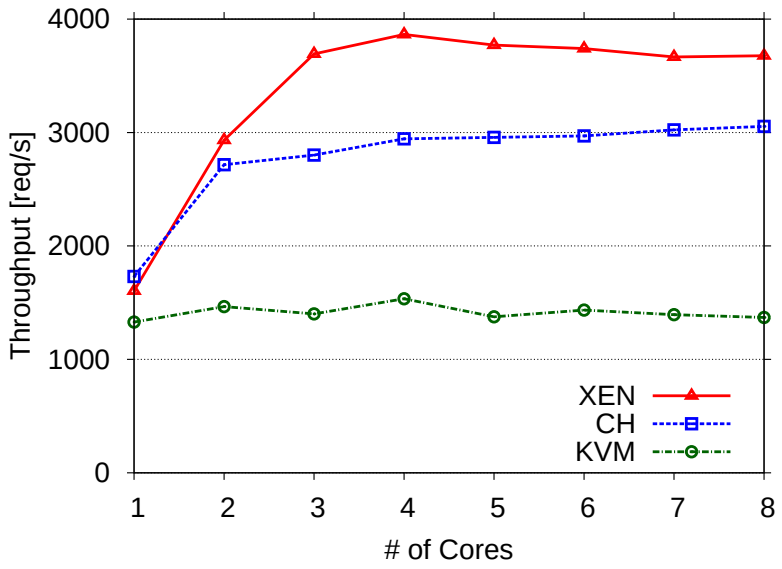
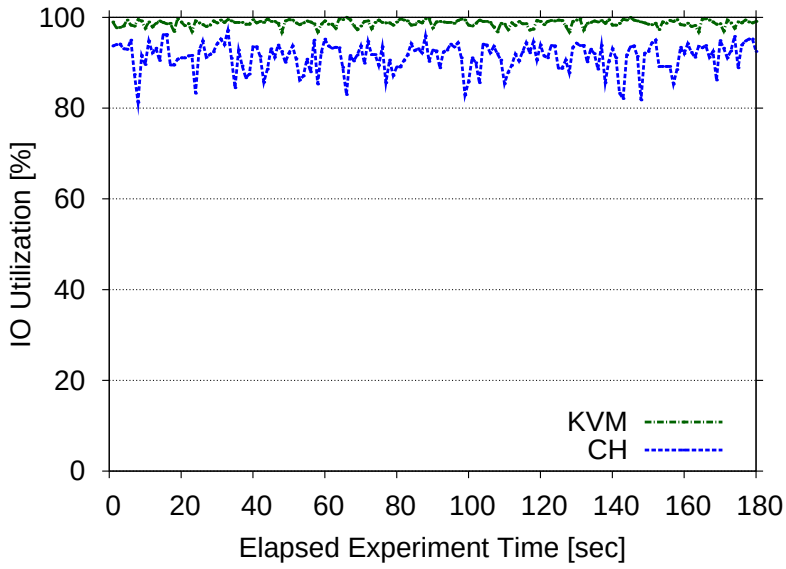


Figure 9: Max throughput comparison among 3 hypervisors when scaling from 1 core to 8 cores; CH achieves at most 123% better performance than KVM and XEN provides up to 32% more throughput than CH.

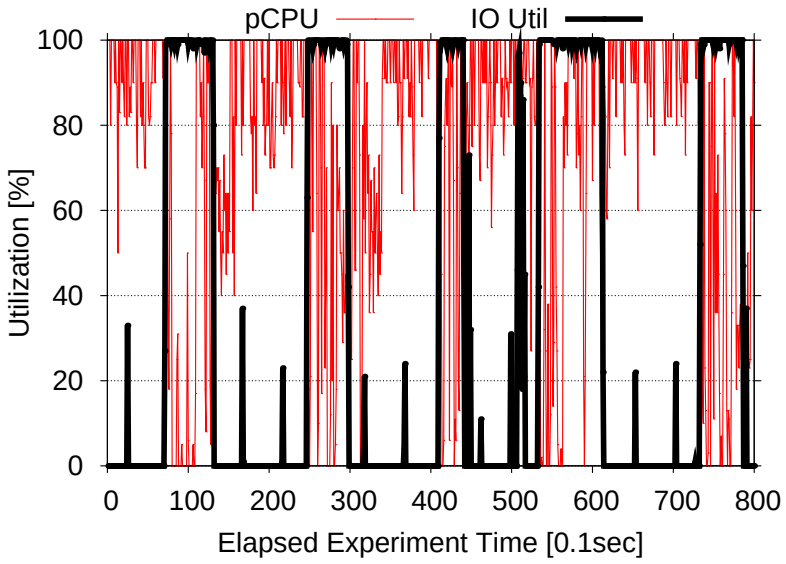
in detail.

The limited intra-MCP horizontal scalability problem can be trace to the different bottlenecks among hypervisors. Under KVM and CH environments, disk I/O was fully saturated and under XEN environment it was due to rapid oscillating bottleneck between CPU and disk I/O. Figure 10(a) depicts the timeline of disk I/O utilization under the 4-core case measured using iostat at one second granularity. This figure shows that under CH and KVM environments, the system I/O is fully saturated. Therefore adding more cores had no effect on system throughput. In the case of XEN, we zoom in to the highly aggregated average of the CPU and I/O utilization through fine-grained analysis. Figure 10(b) illustrates timeline of CPU and I/O utilization under the 4-core case. Both utilization data are measured using Collectl at 100ms granularity. In this figure we observed a negative correlation between CPU utilization and disk I/O utilization which suggests a rapidly oscillating bottleneck between the two resources limiting the system scalability to multi-core processors [59, 90].

Next we investigate the second observation - significant performance differences among hypervisors. In order to understand what has caused such performance differences, we first focus on KVM and CH where both cases showed I/O saturation (see Figure 10(a)).



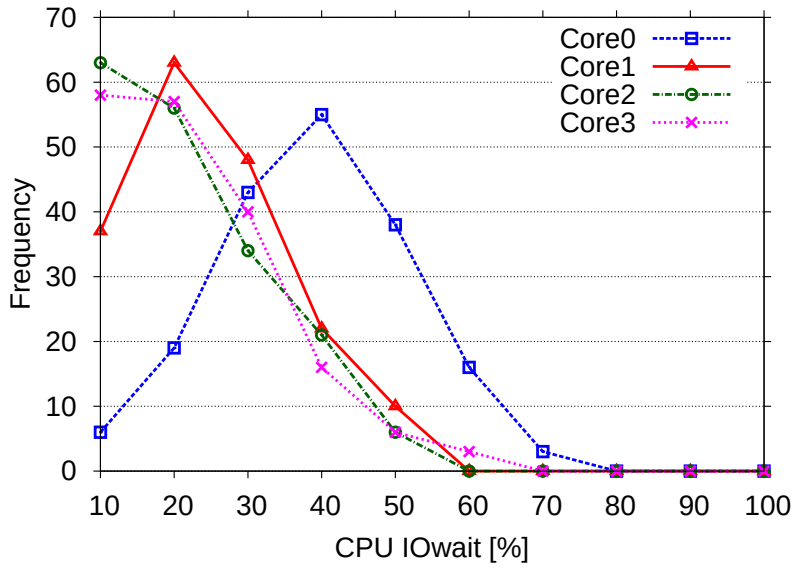
(a) Timeline of disk I/O utilization under CH and KVM configured with 4 cores; Both cases, the disk I/O was the bottleneck limiting system scalability to multi-cores.



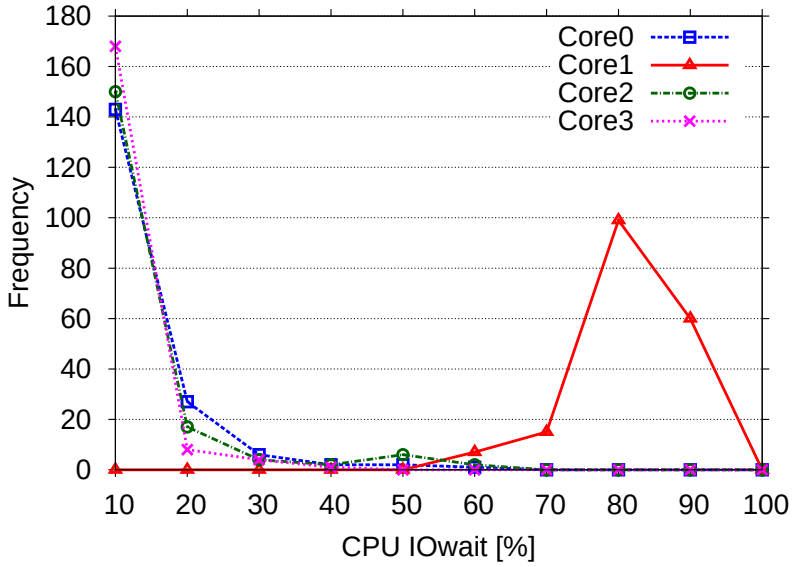
(b) Timeline of CPU and disk I/O under XEN environment; Rapidly oscillating bottleneck between CPU and disk I/O limits system scalability and the I/O utilization indicates that I/O jobs are batched.

Figure 10: Timeline analysis of the hardware resource utilization under 3 hypervisors.

Figure 11(a) and 11(b) illustrates the histogram of CPU IOwait under CH and KVM environments respectively. These CPU IOwait data are collected by sar every 1 second during a 180 second run-time. Figure 11(b) shows that only one core (i.e., Core1) handles most of the I/O under the KVM environment while all other cores showed less than 5% CPU



(a) CPU IOwait Histogram under CH using 4 cores; I/O jobs are distributed to all cores.



(b) CPU IOwait Histogram under KVM using 4 cores; Only one core (i.e., Core1 handles most of the I/O jobs.

Figure 11: CPU IOwait Histogram under 3 hypervisors.

IOwait. This trend was also visible in CPU utilization. Only the core that has high IOwait shows near saturation while others show low utilization. This low utilization in the other cores that are not handling I/O is due to the characteristics of our storage engine (i.e., InnoDB). Since the InnoDB storage engine provides full transactional support, operations

that induce I/O such as updates and inserts trigger various locking mechanisms (e.g., row-lock, table-lock, etc.) which prevent other threads to be processed thus resulting in the low-utilization on other cores. Under CH environment, on the other hand, we observed that I/O is distributed to all cores as depicted in Figure 11(a). This distribution of I/O enables CH to better utilize the underlying multi-core architecture and is the key factor that explains the higher throughput. By distributing I/O to different cores, it allows multiple data manipulation operations that are independent of each other to be executed simultaneously. Consequently this contributes to reducing locking time hence achieves higher utilization by processing more threads under the same amount of time. The average CPU utilization shown in Figure 12 indicates that CH achieves 1.9 times higher utilization than KVM resulting in higher throughput.

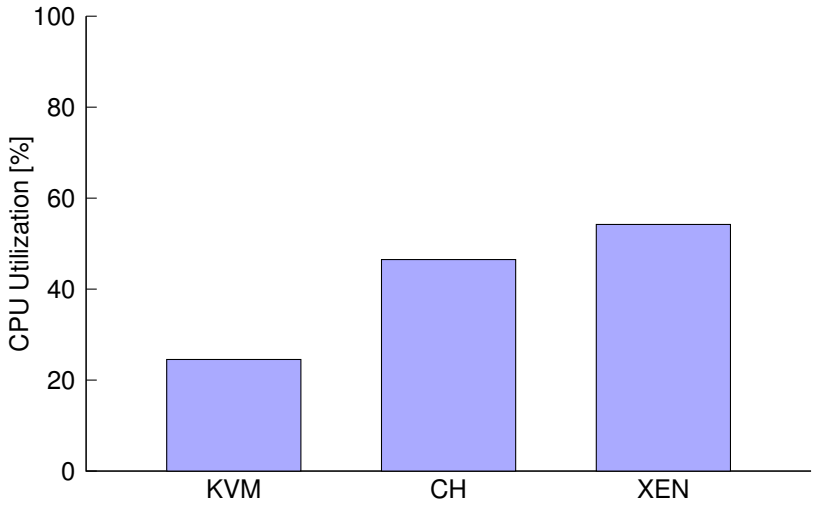


Figure 12: Average CPU Utilization among 3 hypervisors.

Next let's look into XEN case. From the Figure 9 which shows the maximum achievable throughput of the system, we observed that XEN achieves highest throughput among 3 hypervisors. The reason can be traced to its I/O utilization pattern from Figure 10(b). This figure shows that XEN batches I/O tasks inside the memory and writes to the disk intermittently. While XEN is batching I/O requests from the guest domain that is CPU intensive period, increasing the number of cores can relieve CPU bottleneck. As a result, XEN shows best intra-MCP horizontal scalability among 3 hypervisors and achieves 17% higher CPU utilization than CH (see Figure 12) thereby providing better performance.

These results suggests that each hypervisor's strategy of handling write operations in a mixed read/write I/O-intensive workload can have significant impact on n-tier applications performance.

2.6 Related Work

Traditionally, performance analysis in IT systems builds models based on expert knowledge and uses a small set of experimental data to parameterize them [46, 57, 96]. The most popular representative of such models is queuing theory. Queuing networks have been widely applied in many performance prediction methodologies [86, 87]. These approaches are often constrained by their rigid assumptions when handling n-tier systems due to the complex dependencies. As an illustration of significant characteristics that are hard to capture with traditional analysis, consider the significance of context switching towards system performance when a large number of threads is involved [47].

An increasing popularity of virtualization and cloud computing has spawned interesting research on private and public clouds. Barham et al. [19] benchmarked Xen against VMware Workstation and User-Mode Linux, and they showed that Xen outperforms VMware on a range of micro benchmarks and system-wide tests. Clark et al. [29] repeated this performance analysis of Xen in [19] and confirmed the results presented in [19]. They also compared Xen on x86 with IBM zServer and found that the former had a better performance than the latter.

Padala et al. [7] compared Xen and OpenVZ's performance when used for consolidating multi-tiered applications. Their experimental results showed that Xen incurs higher overhead than OpenVZ and average response time can increase by over 400% in Xen and only 100% in OpenVZ as the number of application instances grows from one to four. This can be explained by looking at L2 cache misses; Xen has higher L2 cache misses than OpenVZ. Meanwhile, Adams et al. [7] compared software VMM (binary translation) with hardware-assisted VMM. They showed that software and hardware VMMs both perform well on compute-intensive workloads. However, if workloads include progressively more privileged operations such as context switches, memory mapping, I/O, interrupts and system calls,

both VMMs suffer overheads while software outperforms hardware.

Deshane et al. [34] focused on three aspects of benchmarking Xen and KVM: overall performance, performance isolation, and scalability. They illustrated that Xen has excellent scalability while KVM has substantial problems with guests crashing when a physical node hosts more than four virtual guests. KVM outperforms Xen in isolation. In overall performance tests, Xen has a better performance than KVM on a kernel compile test while KVM outperforms Xen on I/O-intensive tests. Camargos et al. [24] analyzed the performance and scalability of six virtualization technologies (KQEMU, KVM, Linux-VServer, OpenVZ, VirtualBox and Xen) for Linux.

There are multiple research studies on evaluating application performance on multi-core systems with and without virtualization. For example, Xiang et al. [85] analyzed the performance and scalability of para-virtualized VM and hardware-assisted VM on Xen hypervisor (Xen 4.0.0) on a 48-cores shared memory machine using a set of application benchmarks. Their results showed that the tested applications degrade in both performance and scalability on both para-virtualized VM and HVM compared to that on native Linux and they also showed that the main reasons are the additional LLC misses and iTLB misses introduced by virtualization and the idle problem, which is caused by the incompatibility between the Linux idle mechanism (e.g., idle thread) and the Xen idle mechanism (e.g., idle-VM). Similar study by Bryan et al. [89] showed that, due to flow-level parallelism in web server workloads, the number of cache and TLB misses remained nearly constant per byte as the number of cores increased. Likewise, shared cache between cores on the same bus had little effect on performance when compared with unshared cache. Because of flow-level parallelism, there was little data shared between caches.

2.7 Conclusion

In this paper, we studied the horizontal scalability of n-tier application performance within a multi-core processor (MCP) using the n-tier benchmark RUBBoS. We identified mapping of vCPUs to physical cores is an important source of performance variation within the MCP due to its significant influence on cache hit ratio (see Section 2.3). After we eliminated

these variations by fixing the MCP core mapping, we investigated the impact of three mainstream hypervisors (the dominant Commercial Hypervisor, Xen, and KVM) on intra-MCP horizontal scalability and presented some interesting similarities and dissimilarities among the hypervisors. For similarities, we found a non-monotonic scalability trend (throughput increasing up to 4 cores and then decreasing for more than 4 cores) when running a browse-only CPU-intensive workload. This problem can be traced to the inefficient management of last level cache of CPU packages (see Section 2.4). For dissimilarities, we found that each hypervisor's strategy of handling write operations can cause significant performance differences when running a mixed read/write, I/O-intensive workloads (see Section 2.5). Our work suggests that both MCP cache architecture and the choice of hypervisors should be considered as integral components due to its impact on the efficiency and horizontal scalability achievable by n-tier applications.

CHAPTER III

PERFORMANCE INTERFERENCE OF MEMORY THRASHING ON CONSOLIDATED N-TIER APPLICATIONS

This chapter presents second case study of my core thesis research that is the impact of memory thrashing caused by interference among consolidated VMs when we scale horizontally in virtualized cloud environments. First, Section 3.3 illustrates the non-monotonic performance trend as we scale the system horizontally by adding more server VMs within a physical host. We then present extensive measurements and analyses that trace this problem to the hypervisor's IO mechanism which showed large number of page faults (i.e., memory thrashing) caused by interference among consolidated VMs. This memory thrashing induces IO queue overflows in hypervisor's storage stack resulting in very long response time requests due to frequent CPU IOwait although we explicitly mapped virtual to physical resources to remove performance variation [69] and memory was not over-committed. Second, in Section 3.4 we discuss three practical techniques that can reduce the severity of memory thrashing interference or mitigate its effect on the whole system performance. For example, we show that 1) VM migration and 2) memory re-allocation are simple and work well when more computing power is available. We also show that by reducing the concurrency of the system through 3) soft resource re-allocation [94], we can remedy the interference when more resources are unavailable (i.e., VM migration, Memory re-allocation, Soft resource re-allocation) that can mitigate the performance interference caused by memory thrashing.

3.1 Introduction

An increasing number of enterprises are moving their applications to cloud platforms. This rapid growth in computing clouds and datacenters has accelerated the adoption of virtualization technologies, which aims to reduce the cost of operation and maximize profit by consolidating multiple virtual machines (VMs) onto a single physical host. Furthermore, in contrast to traditional consolidation approach that prioritizes performance isolation, recent

studies [48,53] showed that sharing critical resources such as CPU allowed higher utilization and improved overall response time by up to 50%. As a result, the deployment paradigm of the past where each server running on a dedicated platform has evolved into multiple server VMs running simultaneously on a single platform through VM consolidation.

Unfortunately, despite the conceptual simplicity of VM consolidation, leveraging the full potential of this technology has presented significant scalability challenges in practice. One of the most important challenges in clouds and datacenters is the performance interference among consolidated VMs. This has been demonstrated in variety of concrete systems and applications [12, 43, 48, 52, 53, 60, 74] preventing from achieving high level of utilization and barring mission-critical applications due to unpredictable performance. Performance interference is an especially challenging problem for n-tier applications and systems due to the complex dependencies among the hardware and software components in the system [94]. In this work we run extensive experiments to measure and compare the impact of memory thrashing caused by interference among consolidated VMs when we scale horizontally in virtualized cloud environments.

The first and the main contribution of this chapter is the identification of an important source of performance interference: memory thrashing caused by interference among consolidated VMs. For a physical host with 16GB ram, four consolidated VMs each with 2GB memory experienced significant performance loss compared to three consolidated VMs. The performance loss of up to 50% are achieved without any obvious resource bottlenecks. Timeline analysis showed two distinct operational modes within a typical RUBBoS benchmark: the first half of the runtime session showed frequent CPU IOwait causing very long response time (VLRT) requests although the entire dataset is in memory and the workload is read-only. Surprisingly such abnormal CPU IOwait disappeared in the second half resulting moderate utilization when averaged. Subsequent analysis of hypervisor's IO mechanism showed surge of IO requests induced by a large number of page faults (i.e., memory thrashing) although we explicitly mapped virtual to physical resources to remove performance variation [69] and memory was not over-committed.

The second contribution of this chapter is the three remedies that can alleviate memory

thrashing interference. For example, we show that 1) VM migration and 2) memory re-allocation work well when more computing power is available. We also show that by reducing the concurrency of the system through 3) soft resource re-allocation [94], we can remedy the interference when more resources are unavailable.

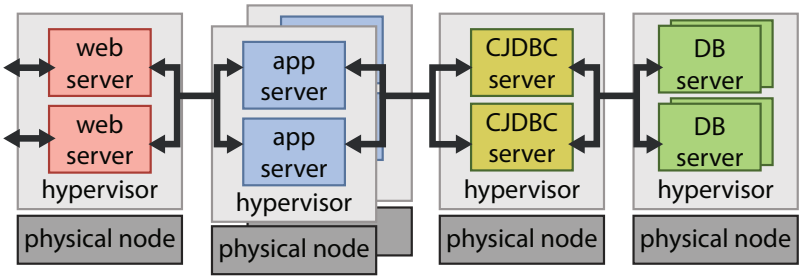
The remainder of this chapter is organized as follows. Section 3.2 presents the description of our experimental setup detailing our n-tier application deployment, profiling environments and tools. Section 3.3 introduces memory thrashing induced by interference among consolidated VMs and its impact on horizontal scalability of n-tier application performance. Section 3.4 discusses three remedies for mitigating the performance interference. Section 3.5 summarizes the related work and Section 3.6 concludes the chapter.

3.2 Experimental Setup

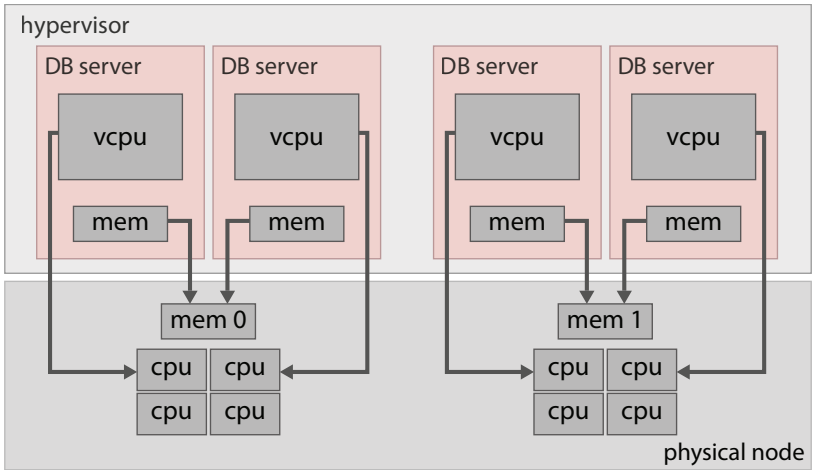
While the horizontal scalability may be evaluated using any type of application, the focus of this paper is n-tier applications with LAMP (Linux, Apache, MySQL, and PHP) implementations. Typically, n-tier applications are organized as a pipeline of servers[1], starting from web servers (e.g., Apache), through application servers (e.g., Tomcat), and ending in database servers (e.g., MySQL). This organization, commonly referred to as n-tier architecture (e.g., 4-tier in Figure 13(a)), serves many important web-facing applications such as e-commerce, customer relationship management, and logistics.

In our experiments, we adopt the RUBBoS n-tier benchmark, based on bulletin board applications such as Slashdot [3]. RUBBoS has been widely used in numerous research efforts due to its real production system significance. The workload includes 24 different interactions such as register user or view story. The benchmark includes two kinds of workload modes: read-only and read/write interaction mixes. Our default experiment trial consists of a three minutes ramp-up, a three minutes run-time, and a 30-second ramp-down. We run the RUBBoS benchmark in a 4-tier system (Figure 13(a)) with workload raging from 1000 up to 35000 users while scaling the system by simply adding more servers to bottleneck tier one at a time. We exploit hardware-assisted VM (HVM) and performance measurements

[1]In this paper, server is used in the sense of computer programs serving client requests. Hardware is referred to as a physical computing node or node for short.



(a) 4-tier application system with 12 servers (i.e., web, application, clustering middleware and database) and five physical hardware nodes in total. Four dedicated DB server VMs are consolidated on a single physical hardware node.



(b) Details of resource mapping between database server VMs and a shared physical hardware node. The VMs' virtual CPUs and memory are explicitly mapped to separate physical CPU packages and memory banks to mitigate interference.

Figure 13: Example of a 4-tier application system deployment, presented as mappings of VMs to physical hosts.

(e.g., CPU utilization) are taken during the run-time period using Sysstat and Collectl at one and 0.05 second granularity respectively. We utilize vendor provided hypervisor monitoring tool to capture low level metrics from the physical host. We exploit ElbaLens - a lightweight request tracing tool, to monitor the trace of transaction executions in our experiments. By time-stamping all messages exchanged between servers at microsecond resolution, it allows us to reconstruct the entire trace of each transaction executed in the system.

At an abstract level, the deployment of n-tier applications in a cloud computing infrastructure can be modeled as a mapping between component servers and physical computing nodes. Figure 13(a) exemplifies our n-tier application deployment with two web server VMs,

Table 2: Summary of experimental setup (i.e., hardware, operating system, software, and virtualization environments).

CPU	Quad Xeon 2.27GHz * 2CPU (8M L3)
Memory	16GB (8GB per Memory Bank)
HDD	SATA, 7200RPM, 500GB
Network I/F	1Gbps
Web Server	HTTPD-2.2.22
App Server	Apache Tomcat-5.5.17
Connector	Tomcat Connectors-1.2.32-src
Clustering Middleware	C-JDBC 2.0.2
DB Server	Mysql-5.5.28-linux2.6-x86_64
Java	JDK-1.6_23
Monitoring Tools	Sysstat, CH monitor, Collectl
Hypervisor	Commercial Hypervisor (CH)
Virtualization Type	Full virtualization (HVM)
Guest OS	RHEL Server 6.3 64-bit
Guest OS Kernel	2.6.32-279.19.1.el6.x86_64

four application server VMs, and two clustering middleware VMs and four database server VMs. For the web-tier, application-tier and clustering middleware-tier, each server VM has two virtual cores (virtual CPUs), 2GB of memory, and 20GB HDD and we consolidated two server VMs on a single physical node using the dominant Commercial Hypervisor[2] (CH) as illustrated in the Figure 13. For the database server VM, each has only one virtual cores and consolidated in a single physical host. In order to mitigate interference between the consolidated VMs, each VM's virtual resources are explicitly mapped (i.e., pin) to separate physical CPU packages and memory banks as shown in Figure 13(b). Through extensive experiments we confirmed that this topology does not introduce any artificial bottlenecks induced by VM consolidation in web-tier, application-tier, and clustering middleware-tier; however these empirical results are omitted here due to space constraints. Other important characteristics of our experimental testbed are summarized in Table 2.

[2]Due to licensing and copy rights issues which prevent publications of performance or comparison data, we mask our choice of commercial virtualization technology. We use commercial hypervisor or CH interchangeably throughout the paper.

3.3 Memory Thrashing Induced by Interference among Consolidated virtual machines (VMs)

One of the straightforward approaches of scaling an n-tier application in modern virtualized cloud platforms is to scale out as known as horizontal scaling (e.g., adding more servers to bottleneck tier). While the improvement of virtualization technologies has facilitated the horizontal scaling through virtual machine (VM) consolidation, we found this approach may hit upper limit.

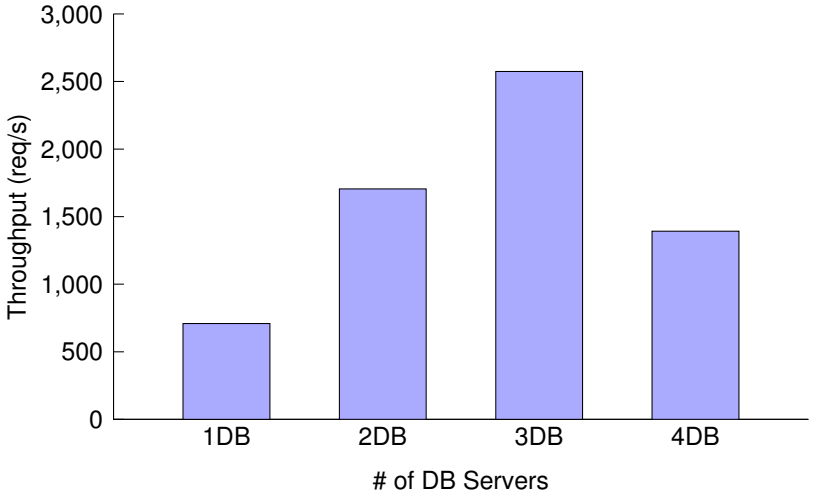


Figure 14: Throughput of the system under 15000 users when scaling from one database server virtual machine to four database server virtual machines consolidated in one physical host; the system shows a non-monotonic scalability trend.

Figure 14 illustrates the performance trend as we scale from one database server virtual machine to four database server virtual machines on a single physical host. Each throughput value used in the figure is captured under a given number of server virtual machine(s) at 15000 users. This result shows that the system was able to scale linearly up to three server virtual machines producing at least 50% more throughput each time; however from three database server virtual machines to four database server virtual machines, the system stops scaling further and the performance deteriorated by nearly 50%. While both three database server virtual machines and four database server virtual machines reported moderate CPU utilization (e.g., 81% and 78% respectively), response time histogram comparison shows clear difference between the two as illustrated in Figure 15.

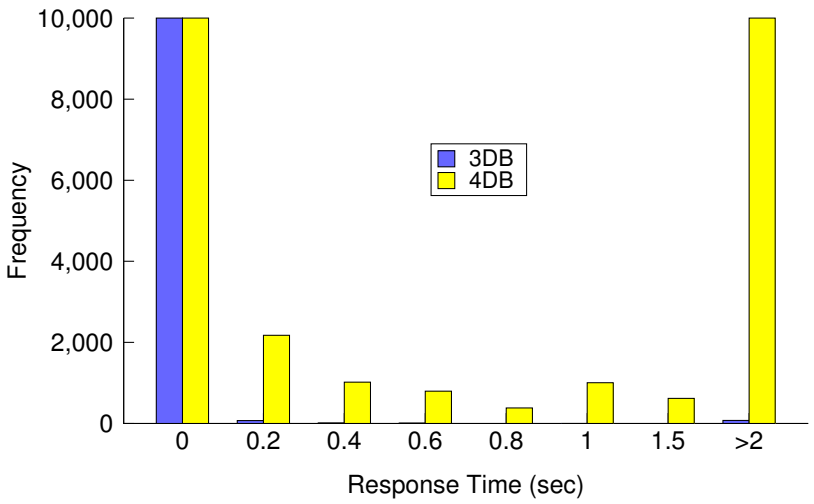


Figure 15: Response time distribution comparison between three database server virtual machines and four database server virtual machines consolidation cases; the four database server virtual machines case shows large number of very long response time (i.e., over 2sec) requests.

Figure 15 compares response time distribution of three database server virtual machines and four database server virtual machines under 15000 workload. The response time bucket size is 200 ms up to one second, 500 ms up to two second and all other requests that took more than two seconds is added into last bucket. In this figure, we found that the response time distribution of three database server virtual machines shows expected long tail distribution, whereas four database server virtual machines shows bi-model distribution with significantly higher number of 'very long response time' (VLRT) requests (i.e., over two seconds).

To investigate the significant increase of the very long response time (VLRT) requests while having more power from the additional server, we break down aggregated CPU utilization into each component. Figure 16 depicts the timeline of CPU utilization and its components under four database server virtual machines consolidation case measured using sar at one second granularity. The green line represents overall CPU utilization, the yellow line represents CPU IOwait, and the blue represents the CPU User. In this graph we observe two distinct operational modes during a typical RUBBoS benchmark experiment. For instance, CPU IOwait dominated over the first part of run-time session saturating the CPU whereas the second part showed no such IO bottleneck and CPU assumes moderate

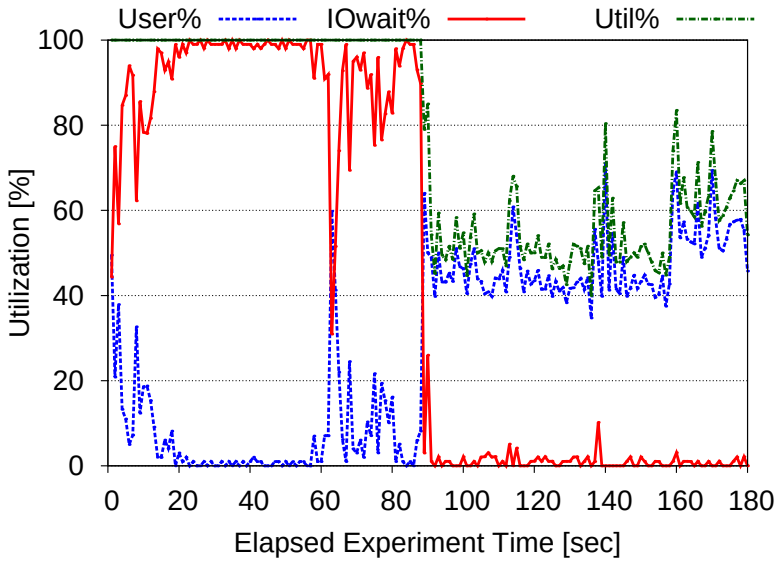


Figure 16: Timeline of CPU utilization breakdown at 15000 users under four database server virtual machines consolidation case; Two distinct operational modes can be observed; Over the first part, IOwait saturates CPU whereas latter part showed no such IO bottleneck.

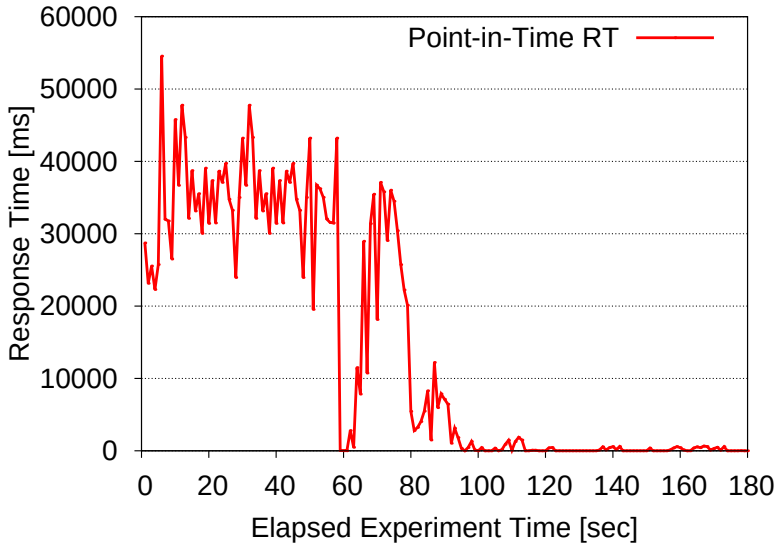


Figure 17: System response time averaged in every one second (Point-in-Time response time); Very Long Response Time (VLRT) requests occurs over the first part and disappears over the second part indicating strong correlation with IO bottleneck (CPU IOwait) shown in the Figure 16.

utilization.

The impact of this IO bottleneck (CPU IOwait) in the first part can be seen clearly through Point-in-Time response time graph shown in Figure 17. Point-in-Time response

time in the context of our work refers to average response time of the requests finished in the given second. In the Figure 17, it is clear that very long response time requests have strong correlation with IO bottleneck shown in Figure 16. It shows that very long response time requests occurs during the first part and disappears over the second part as IO bottleneck mitigates resulting stable system performance. In our work, we categorize these two distinct operational modes as 1) "*Congested Mode*" where we have frequent CPU IOwait causing very long response time (VLRT) requests and 2) "*Normal Mode*" where we have no such IO interference.

Although we scale out within a physical host through virtual machine consolidation, the significant IO interference under four database server virtual machines consolidation case is unexpected. This is due to the fact that we utilized read-only CPU intensive workload and our initial data set size is relatively small (i.e., 300MB), all of which can be loaded into each virtual machine's 2GB of memory. Moreover we explicitly mapped virtual to physical resources in order to remove performance variation [69] and physical host's resources was not over-committed as it still has extra 4 cores, 8GB of memory, and over 300GB of disk space to support hypervisor's operation.

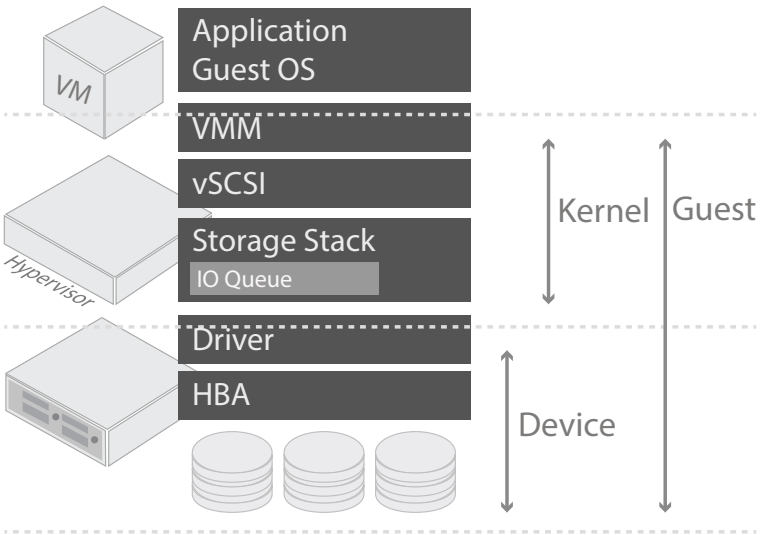


Figure 18: Schematic illustration of IO handling mechanism inside hypervisor kernel and three latency monitoring points.

We further traced the problem into hypervisor's kernel design. Figure 18 depicts schematic illustration of IO request handling mechanism inside the hypervisor kernel and three monitoring points for IO latency measurement. Using the monitoring tool provided by the hypervisor vendor, we captured IO request latency on three layers (i.e., Guest, Kernel, and Device) averaged at two seconds granularity. The guest latency, in our work, refers to total latency as seen from the top of hypervisor and it is sum of kernel latency and device latency. The kernel latency denotes the time an IO request spent waiting inside the IO storage stack within the kernel and the device latency indicate latency coming from the physical hardware. By comparing IO latency of congested mode versus normal mode we found that there is more than two orders of magnitude difference between the two modes as shown in Figure 19.

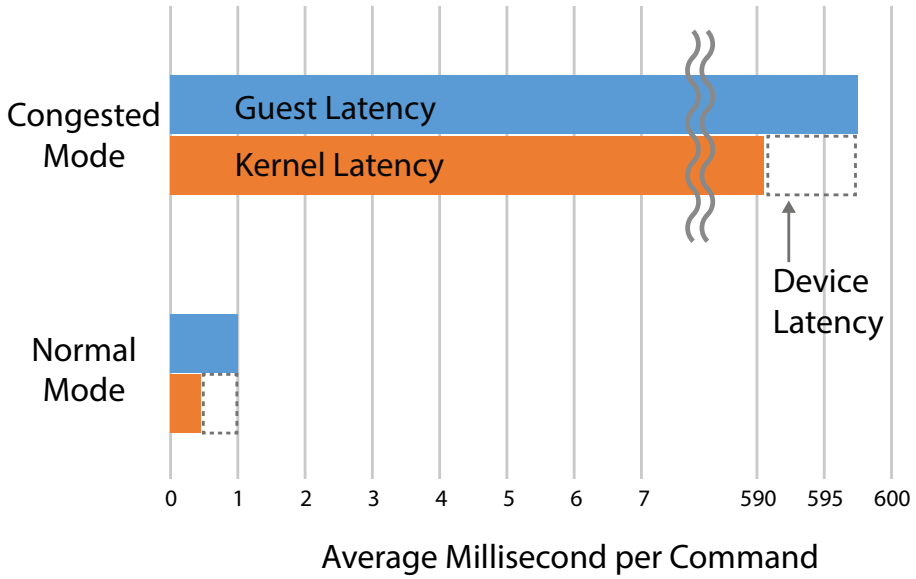


Figure 19: Latency comparison between congested mode and normal mode; congested mode shows more than 500 times higher latency.

Figure 19 illustrates latency comparison between the congested mode and the normal mode. The blue bar represents guest latency that is equivalent to the sum of kernel latency and the device latency. The orange bar denotes kernel latency and the gray bar depicts the device latency. Each latency value is averaged for three minutes run-time session. This figure indicates that there is significant increase in kernel latency dominating the overall

guest latency resulting more than two orders of magnitude difference between the congested mode and the normal mode.

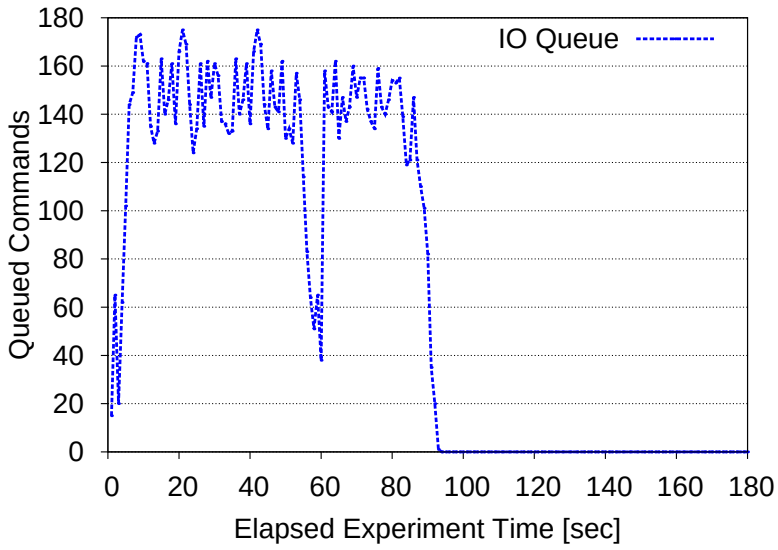


Figure 20: Timeline of IO queue length inside hypervisor's storage stack; significant number of IO commands are queued during congested mode.

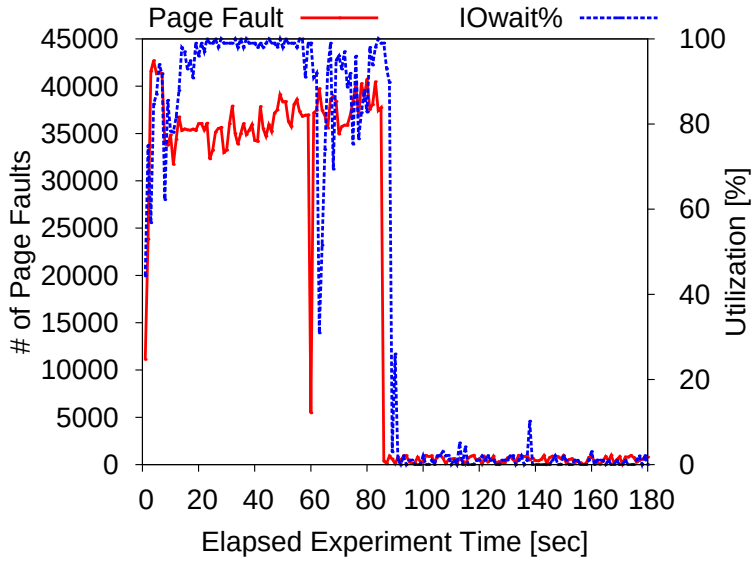


Figure 21: Timeline comparison of page faults and CPU IOwait measured at 1 second interval; large number of page faults occur during congested mode with strong correlation to CPU IOwait and IO Queue length in Figure 20.

To understand the significant increase in kernel latency during congested mode, we monitor the queue inside the hypervisor's IO storage stack (e.g., Figure 18). Figure 20

shows timeline of IO queue length averaged over every two seconds. This figure indicates that significant number of IO commands are queued during congested mode compared to normal mode.

Through the data collection from a large number of experiments and careful analysis, we found that the number of page fault is connected to the large number of queued IO requests during congested mode. Figure 21 illustrate timeline of the number of page fault and CPU IOwait measured at 1-second granularity. In this figure we observed large number of page faults occur during the congested mode and the page faults shows strong correlation to CPU IOwait not to mention the IO queue length shown in Figure 20.

Based on this result we realized that memory thrashing (i.e., page faults) induced by interference among consolidated virtual machines generated large number of IO requests creating a long queue inside the hypervisor which leads to frequent CPU IOwait resulting performance degradation.

3.4 Techniques to Mitigate Performance Interference Induced by Memory Thrashing

In the previous section, we have discussed that memory thrashing can cause significant performance degradation due to its influence on IO commands. In this section we describe techniques that reduce its severity or mitigate its effect on the whole-system performance. We start by describing simple yet effective techniques (i.e., 1. VM Migration, 2. Memory Re-allocation) and end by more sophisticated but economic technique (i.e., 3. Soft Resource Re-Allocation).

One of the most straightforward strategy to solve performance interference among consolidated virtual machines is migrating server virtual machines (VMs) to the additional physical hosts. Since we observed positive performance scalability from one database server virtual machine to three database server virtual machines as shown in Figure 14, we exploit an additional physical machine to migrate two database server virtual machines into the second physical machine. Figure 22 illustrates performance comparison among three database server virtual machines & four database server virtual machines in one physical host and four database server virtual machines in two physical hosts under 15000 users. Figure 22

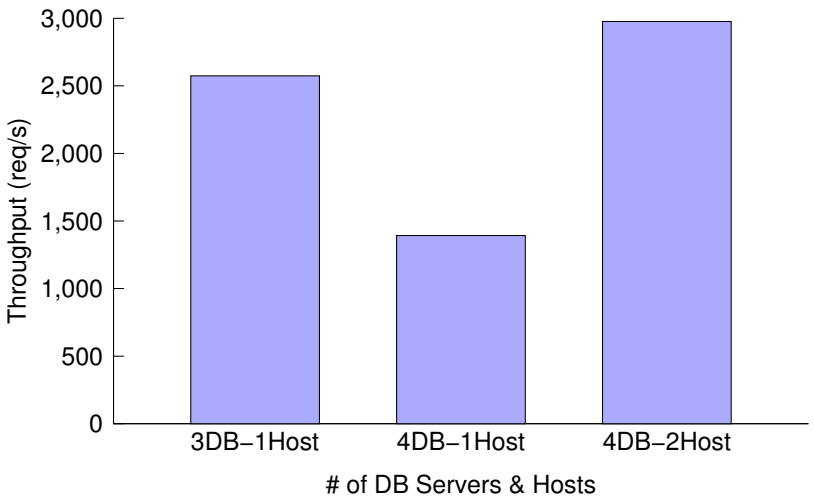


Figure 22: Max throughput comparison among three database server virtual machines in one physical host (3DB-1Host), four database server virtual machines in one physical host (4DB-1Host) and four database server virtual machines in two physical hosts as two database server virtual machines in each host (4DB-2Host); we observe 4DB-2Host achieves two times better performance compared to 4DB-1Host.

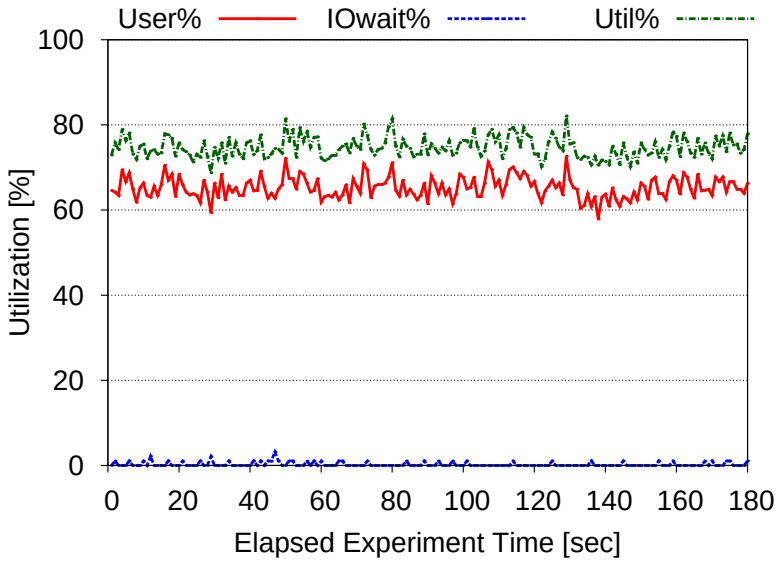


Figure 23: Timeline of CPU utilization and its components under four database server virtual machines in two physical hosts as two database server virtual machines in each host (4DB-2Host); we no longer observe congested mode.

shows migrating two database server virtual machines to an additional physical host produces significant performance gains comapred to all four database server virtual machines consolidated in a single physical host. Moreover we no longer see two distinct operational modes (i.e., congested mode and normal mode) from the timeline of CPU utilization and

its components as shown in Figure 23.

An experienced reader may immediately question the wisdom of migration when comparing three database server virtual machines in one physical host (3DB-1Host) with four database server virtual machines in two physical hosts (4DB-2Host) which appears to be insignificant as shown in Figure 22. This is because when four database server virtual machines run on two physical hosts (4DB-2Host), the bottleneck of the system shifts to upper tiers which limited system scalability. For example the clustering middleware-tier (CJDBC) showed 95% averaged CPU utilization under 4DB-2Host as opposed to 41% under 4DB-1Host.

Figure 24: Max throughput comparison among four database server virtual machines each with 2GB memory in a physical host (4DB-2GB-1Host), four database server virtual machines each with 3GB memory in a physical host (4DB-3GB-1Host) and four database server virtual machines each with 2GB memory in two physical hosts (4DB-2GB-2Host); Allocating more memory to each virtual machine alleviated memory thrashing problem and achieved same throughput as migration.

Another simple technique is memory re-allocation. Since the physical host still has additional 8GB of free memory, we allocate 1GB of memory to each database server virtual machine without over-committing the memory. By increasing memory allocation, we were able to alleviate memory thrashing problem. For example, Figure 24 shows throughput comparison among four database server virtual machines each with 2GB memory (4DB-2GB-1Host) & four database server virtual machines each with 3GB memory in a single physical host (4DB-3GB-1Host), and four database server virtual machines each with 2GB

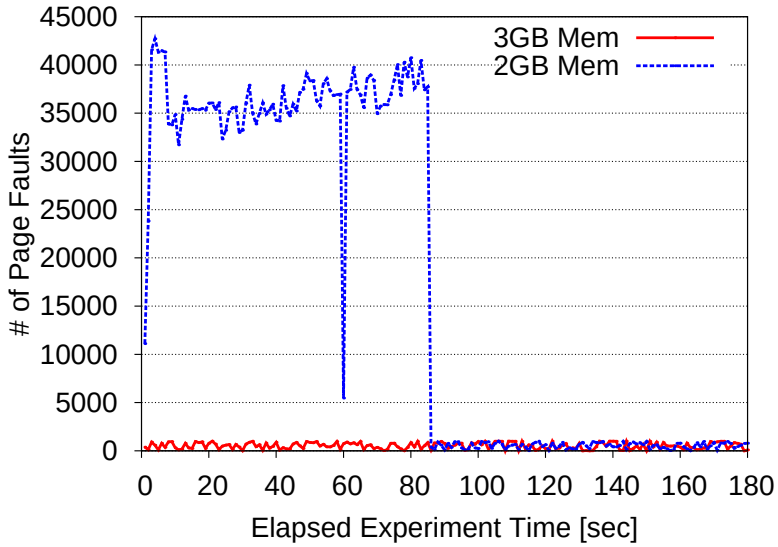
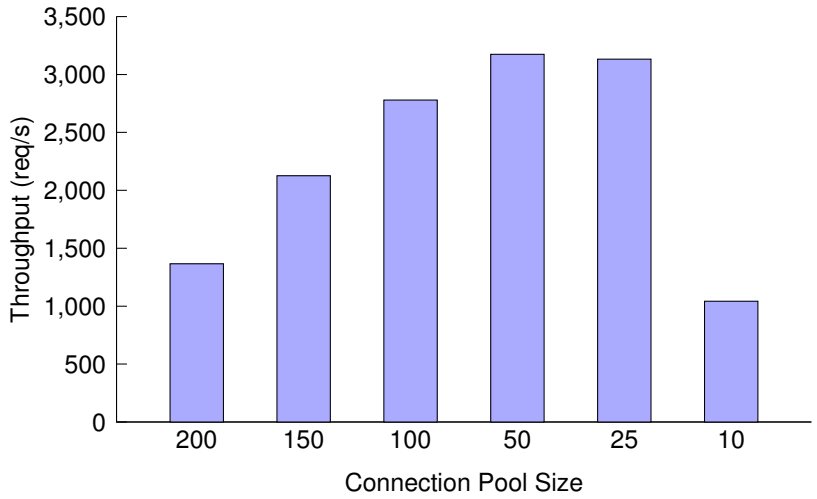


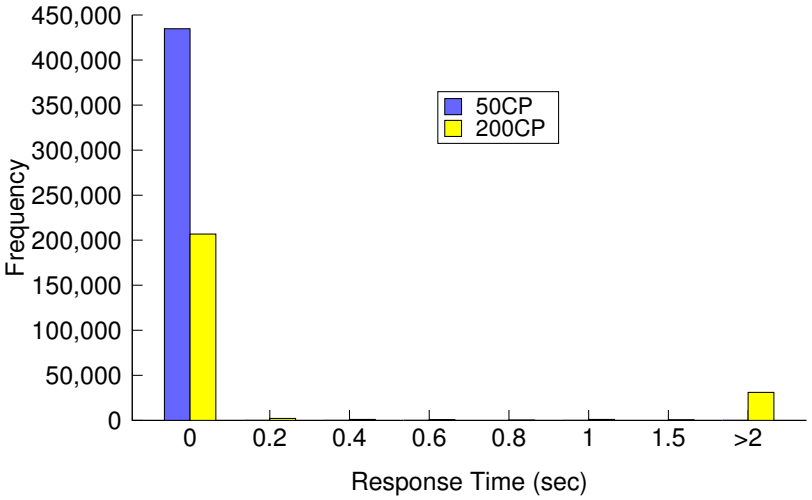
Figure 25: Timeline of page fault comparison between 2GB and 3GB memory allocation; we no longer observe large number of page faults under congested mode.

memory in two physical hosts (4DB-2GB-2Host). This figure not only shows allocating more memory generates significantly higher throughput than initial 2GB allocation but also achieves similar throughput as the migration technique. We also found no IO interference from timeline of CPU utilization similarly to Figure 23 indicating that 1GB of additional memory alleviated IO interference. Moreover we no longer see any notable number of page faults compared to 2GB allocation as shown in Figure 25.

The two techniques introduced so far are straightforward and easy to execute; however more hardware support is often a luxury that administrators cannot afford. For this resource hungry situation, we provide the last technique using soft resource re-allocation. Soft resource allocation in our study refers to system software components that use hardware or synchronize the use of hardware such as thread and connection pool. Previous study conducted by Wang et al. [94] showed that both over allocation and under allocation of soft resources can lead to significant performance loss. Based on this, we start by adjusting connection pool size of apache-tomcat connector in order to reduce the concurrency of the whole system. By reducing the concurrent requests we were able to relieve memory stress creating similar effect as increasing the memory allocation. Figure 26(a) illustrates average throughput of the system as we scale down the connection pool size from initial 200 to 10



(a) Max throughput of the system when scaling down from initial 200 connection pool size to 10 connection pool size; by reducing the concurrency of the system, we can alleviate memory overhead and achieve same throughput as virtual machine migration and memory re-allocation; however too small allocation may cause performance degradation even further.



(b) Response time distribution comparison between 200 connection pool size and 50 connection pool size; the 200 connection pool case shows large number of very long response time (i.e., over 2 seconds) requests.

Figure 26: Analyses of *Soft Resource Allocation* strategy to bypass performance interference

for each tomcat server. In this figure we observed that system throughput increases linearly as we decrease the connection pool size; however from 25 to 10 connection pool size, system throughput deteriorates significantly achieving even less throughput than our initial configuration. In addition we compare the response time distribution of the initial connection

pool size of 200 versus 50 as shown in Figure 26(b). It indicates that 50 connection pool size has 110% higher throughput than the initial 200 using 0.2 seconds as a threshold.

The results of all three techniques suggest that by alleviating the stress on the memory we are able to reduce the severity of IO interference caused by memory thrashing or mitigate its effect on the system performance.

3.5 Related Work

The recent shift in cloud computing model has made VM consolidation in virtualized cloud environments a very active research topic. Many research works model and solve consolidation as a bin-packing optimization problem assuming linear consolidation performance [37, 41, 61, 82]. For example, Ferreto et al. [37] apply linear programming to improve consolidated application performance through dynamic VM migration. Our experimental study of consolidation performance does not invalidate these good results, but our work helps to delimit the applicability of such results that assume linear consolidation performance.

Many previous research efforts have studied the performance interference among consolidated applications [12, 43, 48, 52, 53, 60, 74]. For example, Apparao et al. [12] have characterized and analyzed server consolidation benchmarks. Their experiment results have shown that the performance of any workload suffers considerable loss when it is run in a consolidated environment. Sen et al. [81] investigated the problem of datacenter consolidation with the goal of minimizing the total costs of running a data center. Paul et al. [71] observed performance variations of 25-65% for database servers and 7-40% for file servers. Earlier studies focused on providing perfect isolation among VMs to mitigate the interference. For instance, Gupta et al. [43] introduced ShareGuard to specify the total number of resources consumed in privileged and driver domains in Xen to improve performance isolation. Ye et al. [98] proposed VM level and core level cache awareness optimization methods to further enhance performance isolation among VMs. However recent works showed evidence that sharing is better than isolation due limitations imposed by perfect isolation. For example,

Kanemasa et al. [48] observed sharing critical resources such as CPU allowed better utilization thus improving response time up to 50% compared to isolation scenario. In our work, we contribute to recent effort by identifying one of the important source of interferences under sharing approach.

The latency long tail problem of web applications at moderate levels has been reported greatly over the years [32, 49, 55]. This long-tail latency is a particular concern for n-tier web-facing applications since the problem stems from complex interactions among various system components and not the requests themselves. There has been a lot of work finding a solution on a single server/platform [55], but not on multi-tier systems. Dean et al. [32] described their efforts to bypass the very long response time (VLRT) requests in Google's interactive applications.

Memory thrashing due to page faults is a well known problem to virtualization community. It often occurs when hypervisor is unable to manage memory over-commitment reliably. Large body of studies were carried out by both industry and academia parties. For example, Banerjee et al. [17] conducted comparative analysis of various memory over-commitment methods that are currently implemented on well known hypervisors such as ESX, KVM, Hyper-V, XEN. Gupta et al. [44] demonstrated better use of host memory in Xen through sub-page level page sharing using patching. In our work, we showed that memory thrashing can occur without memory over-commitment and can significantly limit VM consolidation ratio.

3.6 Conclusion

In this paper, we studied the impact of memory thrashing induced by interference among consolidated virtual machines on n-tier application performance. We identified that the memory thrashing caused by virtual machine consolidation is an important source of unpredictable performance due to its significant influence on hypervisors' efficiency in IO management. Concretely, we observed the occurrence of large number of very long response time (VLRT) requests as we scale from three database server virtual machines to four database server virtual machines (Section 3.3). We present three practical remedies

for the memory thrashing problem induced by virtual machine interference, including virtual machine migration, memory re-allocation, and soft resource re-allocation to reduce the severity of interference caused by memory thrashing (Section 3.4). Our work suggests that memory thrashing can occur without memory over-commitment and can significantly limit virtual machine consolidation ratio; however with careful adjustment we are able to bypass or mitigate its impact on n-tier application performance.

CHAPTER IV

AN EMPIRICAL STUDY OF VERY SHORT BOTTLENECKS INDUCED BY MEMORY THRASHING

In the previous chapter, we showed the impact of memory thrashing on horizontal scalability of an n-tier application in virtualized cloud environment through virtual machine consolidation. In this chapter, we show concrete experiment evidence of very short bottleneck induced by memory thrashing and explicitly link this very short bottlenecks with memory thrashing to the very long response time (VLRT) requests. By applying a set of transient event analyses on fine-grained experimental data, we show that very short bottlenecks (from tens to hundreds of milliseconds) can cause queue overflows that propagate through n-tier system, resulting in dropped messages and very long response time (VLRT) requests due to timeouts and re-transmissions. Our study shows that even at moderate CPU utilization levels, very short bottleneck arises due to memory thrashing among consolidated server virtual machines (VMs) resulting significant number of very long response time requests.

4.1 Introduction

Latency long-tail problem (e.g., wide response time fluctuations) of large scale multi-tier applications with moderate system resource utilization levels have been studied both in industry [33] and academia [50, 56, 91, 97]. Every now and then, some requests that usually finish within a few milliseconds may take several seconds. These very long response time (VLRT) requests are extremely hard to study for many reasons. First, the very long response time requests only take a few milliseconds when running by themselves, so the problem is not with the very long response time requests, but arises from the interactions among various system components. Second, when the utilization of the system components are averaged (e.g., average CPU utilization over typical measurement intervals such as several minutes) all of them shows to be far from saturation.

Though our knowledge of the very long response time requests has been limited, practical techniques to bypass the very long response time request problem have been studied in [33]. For instance, applications with read-only characteristic (e.g., web search) can send duplicate requests to independent servers and reduce anticipated response time by choosing the earliest response. These bypassing techniques are valid in some specific cases, contributing to our understanding of the root *causes* for the very long response time requests. However such lack of a detailed understanding of very long response time requests is consistent with the modern data centers' overall low average utilization [84] at around 18%, which is a more practical way to avoid very long response time requests. This current phenomena shows that very long response time requests needs further study to have better knowledge in order to achieve better overall utilization and return on investment in modern data centers.

Earlier studies done by Wang et al. [92] showed that very long response time requests can have very different causes such as CPU dynamic voltage and frequency scaling (DVFS) [92] control at architecture layer, Java garbage collection (GC) at the system software layer, and virtual machine (VM) conslidation at the virtual machine layer through their micro-level event analyses. In addtion to the variety of causes, their results indicated that the non-deterministic nature of very long response time requests makes the events dissimilar at the micro level.

The main contribution of this chapter is a set of transient event analyses of fine-grained experimental data that link very short bottlenecks with memory thrashing to the very long response time (VLRT) requests. The steps of our transient-event analyses are similar to the earlier study conducted by Wang et al. [92]. First we observe very long response time (VLRT) requests while the system is under moderate utilization. Second, Using ElbaLens - a light-weight request tracing tool coupled with fine-grained monitoring tools (a combination of microsecond resolution message timestamping and millisecond system resource sampling), we collect detailed measurement data on a typical n-tier benchmark (RUBBoS [3]) running in virtualized cloud environments. Through careful analyses we discover long request queues are formed in the Apache overflowing TCP buffer, causing re-transmission during that

time. The long queues in Apache are formed because of the queue overflows in saturated downstream tier (i.e., database tier in our study) that propagate through the entire n-tier system. Third, the long queues in database tier are created by very short CPU bottlenecks (CPU IOwait), in which the server CPU becomes saturated for a very short period of time due to frequent CPU IOwait that is CPU waiting IO operation to complete. Fourth, through extensive measurements of an n-tier application benchmark (RUBBoS [3]), we identified that memory thrashing is the root cause associated with the very short bottleneck.

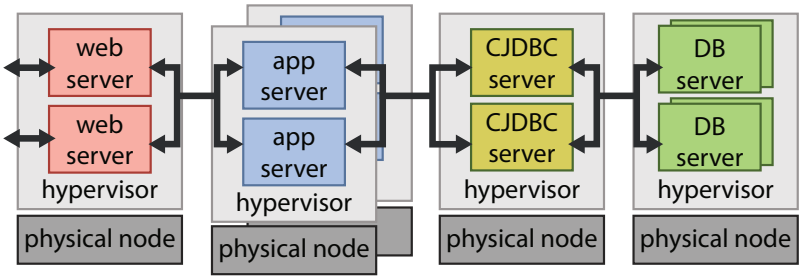
The remainder of this chapter is organized as follows. Section 4.2 presents the description of our experimental setup detailing our n-tier application deployment, profiling environments and tools. Section 4.3 introduces our *Transient Event Analysis* with detailed step-by-step explanation linking very short bottlenecks with memory thrashing to the very long response time (VLRT) requests. Section 4.4 summarizes the related work and Section 4.5 concludes the chapter.

4.2 Experimental Setup

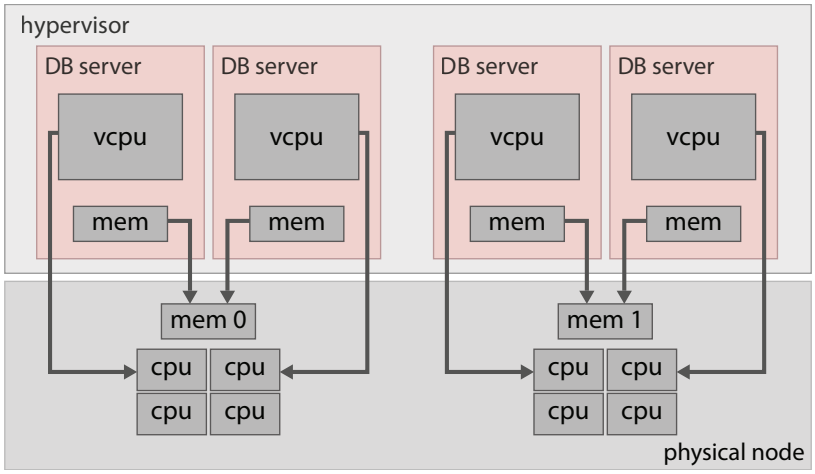
While the horizontal scalability may be evaluated using any type of application, the focus of this paper is n-tier applications with LAMP (Linux, Apache, MySQL, and PHP) implementations. Typically, n-tier applications are organized as a pipeline of servers[1], starting from web servers (e.g., Apache), through application servers (e.g., Tomcat), and ending in database servers (e.g., MySQL). This organization, commonly referred to as n-tier architecture (e.g., 4-tier in Figure 27(a)), serves many important web-facing applications such as e-commerce, customer relationship management, and logistics.

In our experiments, we adopt the RUBBoS n-tier benchmark, based on bulletin board applications such as Slashdot [3]. RUBBoS has been widely used in numerous research efforts due to its real production system significance. The workload includes 24 different interactions such as register user or view story. The benchmark includes two kinds of workload modes: read-only and read/write interaction mixes. Our default experiment trial consists of a three minutes ramp-up, a three minutes run-time, and a 30-second ramp-down.

[1]In this paper, server is used in the sense of computer programs serving client requests. Hardware is referred to as a physical computing node or node for short.



(a) 4-tier application system with 12 servers (i.e., web, application, clustering middleware and database) and five physical hardware nodes in total. Four dedicated DB server VMs are consolidated on a single physical hardware node.



(b) Details of resource mapping between database server VMs and a shared physical hardware node. The VMs' virtual CPUs and memory are explicitly mapped to separate physical CPU packages and memory banks to mitigate interference.

Figure 27: Example of a 4-tier application system deployment, presented as mappings of VMs to physical hosts.

We run the RUBBoS benchmark in a 4-tier system (Figure 27(a)) with workload raging from 1000 up to 35000 users while scaling the system by simply adding more servers to bottleneck tier one at a time. We exploit hardware-assisted VM (HVM) and performance measurements (e.g., CPU utilization) are taken during the run-time period using Sysstat and Collectl at one and 0.05 second granularity respectively. We utilize vendor provided hypervisor monitoring tool to capture low level metrics from the physical host. We exploit ElbaLens - a lightweight request tracing tool, to monitor the trace of transaction executions in our experiments. By time-stamping all messages exchanged between servers at microsecond resolution, it allows us to reconstruct the entire trace of each transaction executed in the system.

Table 3: Summary of experimental setup (i.e., hardware, operating system, software, and virtualization environments).

CPU	Quad Xeon 2.27GHz * 2CPU (8M L3)
Memory	16GB (8GB per Memory Bank)
HDD	SATA, 7200RPM, 500GB
Network I/F	1Gbps
Web Server	HTTPD-2.2.22
App Server	Apache Tomcat-5.5.17
Connector	Tomcat Connectors-1.2.32-src
Clustering Middleware	C-JDBC 2.0.2
DB Server	Mysql-5.5.28-linux2.6-x86_64
Java	JDK-1.6_23
Monitoring Tools	Sysstat, CH monitor, Collectl
Hypervisor	Commercial Hypervisor (CH)
Virtualization Type	Full virtualization (HVM)
Guest OS	RHEL Server 6.3 64-bit
Guest OS Kernel	2.6.32-279.19.1.el6.x86_64

At an abstract level, the deployment of n-tier applications in a cloud computing infrastructure can be modeled as a mapping between component servers and physical computing nodes. Figure 27(a) exemplifies our n-tier application deployment with two web server VMs, four application server VMs, and two clustering middleware VMs and four database server VMs. For the web-tier, application-tier and clustering middleware-tier, each server VM has two virtual cores (virtual CPUs), 2GB of memory, and 20GB HDD and we consolidated two server VMs on a single physical node using the dominant Commercial Hypervisor[2] (CH) as illustrated in the Figure 27. For the database server VM, each has only one virtual cores and consolidated in a single physical host. In order to mitigate interference between the consolidated VMs, each VM's virtual resources are explicitly mapped (i.e., pin) to separate physical CPU packages and memory banks as shown in Figure 27(b). Through extensive experiments we confirmed that this topology does not introduce any artificial bottlenecks induced by VM consolidation in web-tier, application-tier, and clustering middleware-tier; however these empirical results are omitted here due to space constraints. Other important

[2]Due to licensing and copy rights issues which prevent publications of performance or comparison data, we mask our choice of commercial virtualization technology. We use commercial hypervisor or CH interchangeably throughout the paper.

characteristics of our experimental testbed are summarized in Table 3.

4.3 Very Long Response Time (VLRT) Requests caused by Memory Thrashing

In the previous chapter, we showed that memory thrashing induced by interference among consolidated virtual machines produced significant performance losses due to the frequent CPU IOwait. In this chapter we use micro level analysis that exploits the fine-grained measurement data collected in RUBBoS experiments to establish the link between very long response time requests shown in the previous chapter (e.g., Figure 17, 15) to the memory thrashing (i.e., page faults) in the database server tier. More specifically, we utilized ElbaLens, a lightweight tracing tool for performance debugging of web services, to timestamp all messages exchanged between servers at microsecond resolution. By recording the precise arrival and departure time of each request for all servers, we were able to determine how long each request spends in each tier. In addition system resources (e.g., CPU) are monitored by CollectL at time granularity of 50ms to capture the very short bottleneck window. The steps of our transient-event analyses closely follow the earlier study conducted by Wang et al. [92] and the events are shown in the timeline graphs where X-axis represents the time elapsed during the experiment at 50ms granularity.

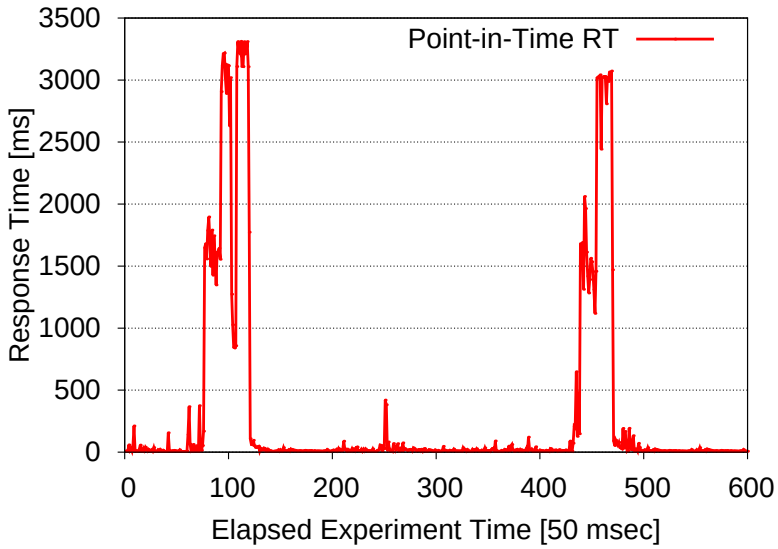


Figure 28: Fine-grained response time measured at 50ms interval. Large response time fluctuation with its peaks correspond to the peaks of queued requests in Figure 29.

In order to reduce the noise on the data that is inherent in fine-grained monitoring data, we ran a light-weight micro-benchmark within the typical n-tier application benchmark RUBBoS under sterile environment (i.e., no hardware bottlenecks in any tier). Our micro-benchmark randomly requests 1.2GB of memory during runtime session of our experiment and releases it as soon as it recognizes all of the memory it had requested. To measure the impact of the micro-benchmark in fine detail, we utilized ElbaLens, a lightweight tracing tool for performance debugging of web services, to timestamp all messages exchanged between servers at microsecond resolution. In addition system resource utilization is monitored through CollectL at time granularity of 50ms to capture the very short bottleneck window.

In the first step of transient-event analysis (i.e., very long response time in short period), we utilize fine-grained monitoring data to determine at which time point client requests are taking seconds to finish instead of the normally expected milliseconds response time. Figure 28 is a point-in-time response time graph over 30 seconds during 3min run-time session. The x-axis of figure 28 is a timeline at 50ms interval, showing clusters of very long response time requests are tightly grouped within couple of seconds. Figure 28 shows two peak/clusters of very long response time requests during a 30-second period of a RUBBoS experiment (workload 9000 clients). Outside of these peaks, the response time of all requests return within milliseconds, consistent with the average CPU utilization among component servers being equal to or lower than 73%.

In the second step of transient-event analysis (queue amplification), we first show that dropped message packets are likely the cause of VLRT requests. To make this connection, we first determine which events are being queued in each server. In an n-tier system, we say that a request is waiting in a queue at a given tier when its request packet has arrived and a response has not been returned to an upstream server or client. This situation is the n-tier system equivalent of having a program counter entering that server but not yet exited. Using the same timeframe of Figure 28, we plot the request queue length in the Apache server in Figure 29. In Figure 29 we discover long request queues are formed in the Apache which strongly suggests that requests overflowed TCP buffer, causing re-transmission during that time [92].

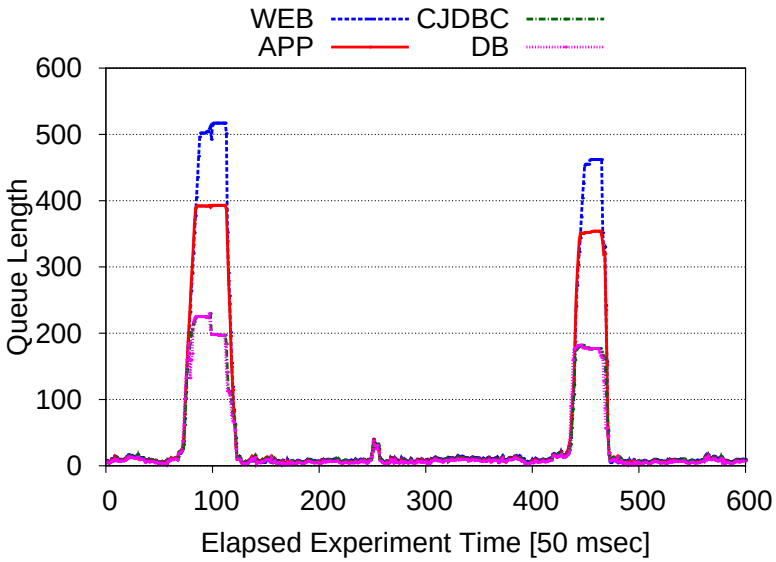


Figure 29: Fine-grained queued request length of each tier counted at every 50ms time window. Queue peaks in each tier coincide with the queue peaks in downstream tiers, suggesting push-back wave from database tier all the way to WEB tier.

Since Apache itself is not a bottleneck (none of the Apache resources is a bottleneck), we can infer the long queues in Apache are formed because of the queue overflows in saturated downstream tier. Therefore we continue the per-server queue analysis by integrating and comparing the requests queued in Apache with the requests queued in Tomcat, with the requests queued in CJDBC (clustering middleware), and with the requests queued in MySQL (database). The two major peak/clusters in Figure 29 show the queued requests in all four tiers from database tier to clustering middleware tier to application server tier to web server tier. This near-perfect coincidence of (very regular and very short) queuing episodes suggests that it is not by chance, but somehow Database tier may have contributed to the queued requests in upstream tiers all the way to Apache.

Let us consider more generally the situation in n-tier systems where queuing in a downstream server is associated with queuing in the upstream server. In client/server n-tier systems, a client request is sent downstream for processing, with a pending thread in the upstream server waiting for the response. If the downstream server encounters internal processing delays, two things happen. First, the downstream server's queue grows. Second, the number of matching and waiting threads in the upstream server also grows due to the

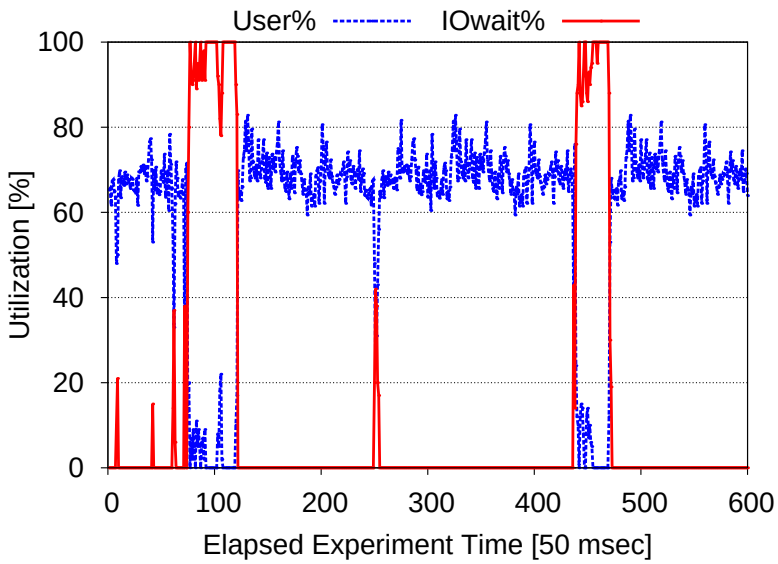


Figure 30: Fine-grained CPU utilization captured at 50ms granularity. Transient CPU saturations by CPU IOwait in database tier match well with the queue peaks in Figure 29.

lack of responses from downstream. Similar queuing episodes studied earlier [92] named this phenomenon as *push-back wave*. The result of the second step in transient event analysis is the series of connections between VLRT to dropped message packets to long queue in Apache to queuing in Tomcat to queuing in CJDBC and to queuing in MySQL due to MySQL saturation.

In the third step of transient event analysis (very short bottleneck), we link the database tier queuing with very short bottlenecks in which CPU becomes saturated for a short period time. Figure 30 depicts the timeline of database tier CPU utilization breakdown measured at 50ms interval within the same timeframe as figure 29. We can see two peaks where database tier CPU utilization become fully saturated by CPU IOwait for a short period of time. This short period of CPU saturation is the very short bottleneck that causes database tier queuing in figure 29 and through pushback wave to CJDBC tier queue, then to Tomcat tier queue, and finally to Apache tier queue as we have seen in figure 29. In addition, the fact that CPU saturation is caused by CPU IOwait not by CPU User indicates the source of this short bottleneck is IO bound.

The fourth step of the transient event analysis (root cause) is the associating IO bound short CPU bottlenecks to page fault episodes. Figure 31 shows the timeline of page faults

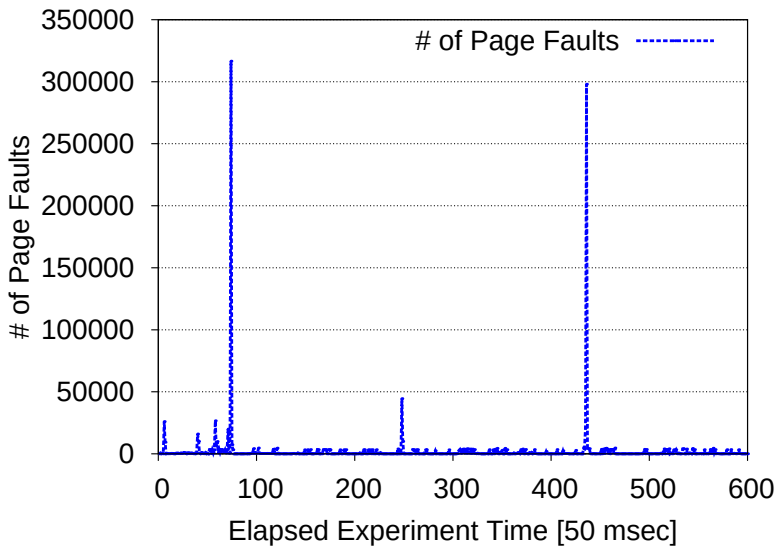


Figure 31: Fine-grained number of page faults captured at every 50ms timeframe. The peaks of the page faults in database tier coincide with the transient CPU saturation of the database tier in Figure 30.

measured at 50ms granularity under the same timeframe as figure 30. In this figure we observe two major peaks of the page faults occurs with near perfect correlation to IO bound short CPU bottleneck found in figure 30. This step shows that sudden surge in page faults has caused frequent CPU IOwait resulting very short bottlenecks.

In summary, the 4 steps of transient event analysis show the very long response time requests in figure 28 are due to the IO bound short CPU bottleneck induced by memory thrashing:

1. Transient events: Very long response time (VLRT) requests are observed while the system is under moderate utilization. (Figure 28).

2. Queue amplification: Very long response time requests coincide with long request queues in the Apache server that causes dropped packets and TCP retransmission. The long queues in Apache are caused by push-back waves downstream tiers, where similar long queues form at the same time (Figure 29).

3. Very short bottlenecks: long queues in MySQL tier (Figure 29) are created by very short bottlenecks (Figure 30), in which the MySQL CPU becomes saturated with

CPU IOwait for a short period of time (about 300 milliseconds).

4. Root cause: The very short bottlenecks coincide near perfectly with page fault episodes (Figure 31).

4.4 Related Work

Latency has received increasing attention in the evaluation of quality of service provided by computing clouds and data centers [13, 70, 79, 90, 91]. Specifically, the long-tail latency is of particular concern for mission-critical web-facing applications [9, 10, 33, 56, 97]. On the solution side, Dean et al. [33] described their efforts to mitigate tail latency in Google's interactive applications. These bypass techniques are effective in specific applications or domains, contributing to an increasingly acute need to improve our understanding of the general causes for the VLRT requests.

Aggregated statistical analyses over fine-grained monitored data have been used to infer the appearance and causes of long-tail latency [30, 56, 90]. Li et al. [56] measure and compare the changes of latency distributions to study hardware, OS, and concurrency-model induced causes of tail latency in typical web servers executing on multi-core machines. Wang et al. [90] propose a fine-grained correlation analysis between a server's throughput and concurrent jobs in the server to infer the server's real-time performance state. Cohen [30] use a class of probabilistic models to correlate system-level metrics and threshold values with high-level performance states. Our work leverages the fine-grain data, but we go further in using micro-level timeline event analysis to link the various causes to VLRT requests.

Our work makes heavy use of data from fine-grained monitoring and profiling tools [4,5]. Related techniques have been proposed to help detect a performance problem and identify symptoms associated with the problem [21, 25, 56, 72, 78]. For example, Collectl [4] provides the ability to monitor a broad set of system level metrics such as CPU and I/O operations at millisecond-level granularity. Chopstix [21] continuously collects profiles of low-level OS events (e.g., scheduling, L2 cache misses, page allocation, locking) at the granularity of executables, procedures and instruction. Li et al. [56] propose a fine-grained timestamping technique to measure how much time a request spends in different parts of the server OS.

We use these tools when applicable.

Techniques based on end-to-end request-flow tracing have been proposed for performance anomaly diagnosis [8, 15, 26, 36, 77, 80], but usually for more stable and longer phenomena. X-ray [15] instruments binaries as applications execute and uses dynamic information flow tracking to estimate the likelihood that a block was executed due to each potential root cause for the performance anomaly. Fay [36] provides dynamic tracing through use of run-time instrumentation and distributed aggregation within machines and across clusters for windows platform. Aguilera et al. [8] infer causal paths between component servers in a distributed system and attribute delays to specific nodes. Pip [77] detects anomalous requests by comparing request-flows from actual behaviors with developer-expected behaviors. Spectroscope [80] is similar to Pip, but Spectroscope compares request-flows between "problem" periods and "non-problem" periods for identifying anomalous requests.

4.5 Conclusion

Applying a transient event analysis on extensive experimental data collected from fine-grain monitoring of n-tier application benchmarks, we demonstrate that the very long response time (VLRT) requests may arise from memory thrashing among consolidated virtual machines (VMs). These phenomena can be modeled and described as *"very short bottlenecks"*, very short periods of time (tens to hundreds of milliseconds) in which the CPU is saturated. The transient event analysis shows that the very long response time requests are coincidental to very short bottlenecks in database servers caused by memory thrashing, which in turn propagate and amplify queuing in upstream servers, quickly leading to TCP buffer overflow and frequent request re-transmission, resulting very long response time requests of several seconds.

CHAPTER V

RELATED WORK

Traditionally, performance analysis in IT systems builds models based on expert knowledge and uses a small set of experimental data to parameterize them [46, 57, 96]. The most popular representative of such models is queuing theory. Queuing networks have been widely applied in many performance prediction methodologies [86, 87]. These approaches are often constrained by their rigid assumptions when handling n-tier systems due to the complex dependencies. As an illustration of significant characteristics that are hard to capture with traditional analysis, consider the significance of context switching towards system performance when a large number of threads is involved [47].

An increasing popularity of virtualization and cloud computing has spawned interesting research on private and public clouds. Barham et al. [19] benchmarked Xen against VMware Workstation and User-Mode Linux, and they showed that Xen outperforms VMware on a range of micro benchmarks and system-wide tests. Clark et al. [29] repeated this performance analysis of Xen in [19] and confirmed the results presented in [19]. They also compared Xen on x86 with IBM zServer and found that the former had a better performance than the latter.

Padala et al. [7] compared Xen and OpenVZ's performance when used for consolidating multi-tiered applications. Their experimental results showed that Xen incurs higher overhead than OpenVZ and average response time can increase by over 400% in Xen and only 100% in OpenVZ as the number of application instances grows from one to four. This can be explained by looking at L2 cache misses; Xen has higher L2 cache misses than OpenVZ. Meanwhile, Adams et al. [7] compared software VMM (binary translation) with hardware-assisted VMM. They showed that software and hardware VMMs both perform well on compute-intensive workloads. However, if workloads include progressively more privileged operations such as context switches, memory mapping, I/O, interrupts and system calls,

both VMMs suffer overheads while software outperforms hardware.

Deshane et al. [34] focused on three aspects of benchmarking Xen and KVM: overall performance, performance isolation, and scalability. They illustrated that Xen has excellent scalability while KVM has substantial problems with guests crashing when a physical node hosts more than four virtual guests. KVM outperforms Xen in isolation. In overall performance tests, Xen has a better performance than KVM on a kernel compile test while KVM outperforms Xen on I/O-intensive tests. Camargos et al. [24] analyzed the performance and scalability of six virtualization technologies (KQEMU, KVM, Linux-VServer, OpenVZ, VirtualBox and Xen) for Linux.

There are multiple research studies on evaluating application performance on multi-core systems with and without virtualization. For example, Xiang et al. [85] analyzed the performance and scalability of para-virtualized VM and hardware-assisted VM on Xen hypervisor (Xen 4.0.0) on a 48-cores shared memory machine using a set of application benchmarks. Their results showed that the tested applications degrade in both performance and scalability on both para-virtualized VM and HVM compared to that on native Linux and they also showed that the main reasons are the additional LLC misses and iTLB misses introduced by virtualization and the idle problem, which is caused by the incompatibility between the Linux idle mechanism (e.g., idle thread) and the Xen idle mechanism (e.g., idle-VM). Similar study by Bryan et al. [89] showed that, due to flow-level parallelism in web server workloads, the number of cache and TLB misses remained nearly constant per byte as the number of cores increased. Likewise, shared cache between cores on the same bus had little effect on performance when compared with unshared cache. Because of flow-level parallelism, there was little data shared between caches.

The recent shift in cloud computing model has made VM consolidation in virtualized cloud environments a very active research topic due to its practical interest. Many research works model and solve consolidation as a bin-packing optimization problem assuming linear consolidation performance [37, 41, 61, 82]. For example, Ferreto et al. [37] apply linear programming to improve consolidated application performance through dynamic VM migration. Our experimental study of consolidation performance does not invalidate these

good results, but our work helps to delimit the applicability of such results that assume linear consolidation performance.

The implications of virtualization and consolidation are that multiple VMs contend for shared resources on each platform. As such there has been a lot of work published about performance interference among consolidated applications [12,43,48,52,53,60,74]. For example, Apparao et al. [12] have characterized and analyzed server consolidation benchmarks. Their experiment results have shown that the performance of any workload suffers considerable loss when it is run in a consolidated environment. Sen et al. [81] investigated the problem of datacenter consolidation with the goal of minimizing the total costs of running a data center. Paul et al. [71] observed performance variations of 25-65% for database servers and 7-40% for file servers. Earlier studies focused on providing perfect isolation among VMs to mitigate the interference. For instance, Gupta et al. [43] introduced ShareGuard to specify the total number of resources consumed in privileged and driver domains in Xen to improve performance isolation. Ye et al. [98] proposed VM level and core level cache awareness optimization methods to further enhance performance isolation among VMs. However recent works showed evidence that sharing is better than isolation due limitations imposed by perfect isolation. For example, Kanemasa et al. [48] observed sharing critical resources such as CPU allowed better utilization thus improving response time up to 50% compared to isolation scenario. In our work, we contribute to recent effort by identifying one of the important source of interferences under sharing approach through horizontal scalability of n-tier system.

Memory thrashing due to page faults is a well known problem to virtualization community. It often occurs when hypervisor is unable to manage memory over-commitment reliably. Large body of studies were carried out by both industry and academia parties. For example, Banerjee et al. [17] conducted comparative analysis of various memory over-commitment methods that are currently implemented on well known hypervisors such as ESX, KVM, Hyper-V, XEN. Gupta et al. [44] demonstrated better use of host memory in Xen through sub-page level page sharing using patching. In our work, we showed that memory thrashing can occur without memory over-commitment and can significantly limit

VM consolidation under the moderate system utilization levels.

The latency long tail problem of large-scale distributed applications at moderate levels has been reported greatly over the years [32, 49, 55]. This long-tail latency is a particular concern for n-tier web-facing applications since the problem stems from complex interactions among various system components and not the requests themselves. There has been a lot of work finding a solution on a single server/platform [55], but not on multi-tier systems. Dean et al. [32] described their efforts to bypass the very long response time (VLRT) requests in Google's interactive applications.

Latency has received increasing attention in the evaluation of quality of service provided by computing clouds and data centers [13, 70, 79, 90, 91]. Specifically, the long-tail latency is of particular concern for mission-critical web-facing applications [9, 10, 33, 56, 97]. On the solution side, Dean et al. [33] described their efforts to mitigate tail latency in Google's interactive applications. These bypass techniques are effective in specific applications or domains, contributing to an increasingly acute need to improve our understanding of the general causes for the VLRT requests.

The potential causes for the performance problem of web applications have been studied extensively in many previous research. Dean et al. [33] outlined several potential causes for the tail latency problem of Google's large scale interactive applications. Examples include, but not limit to shared resources (such as CPU cores, processor caches) by different applications running on top of the same set of machines, background daemons, maintenance activities, and several hardware trends such as power limits for modern CPUs, garbage collection for solid-state storage devices, and power-saving modes in many types of devices. Software mis-configuration for the failure of distributed systems have been studied in [15,16, 65]. Workloads characteristics such as burstiness or request type mix-ratio for the impact of system performance have been studied in [22, 28, 40, 45, 62–64, 83]. In addition, soft resource (e.g., threads, database connections) allocation has been discussed as an important source for unpredictable performance in [6, 20, 35, 39, 58, 66–68, 75, 95, 99].

Aggregated statistical analyses over fine-grained monitored data have been used to infer

the appearance and causes of long-tail latency [30, 56, 90]. Li et al. [56] measure and compare the changes of latency distributions to study hardware, OS, and concurrency-model induced causes of tail latency in typical web servers executing on multi-core machines. Wang et al. [90] propose a fine-grained correlation analysis between a server's throughput and concurrent jobs in the server to infer the server's real-time performance state. Cohen [30] use a class of probabilistic models to correlate system-level metrics and threshold values with high-level performance states. Our work leverages the fine-grain data, but we go further in using micro-level timeline event analysis to link the various causes to VLRT requests.

Our work makes heavy use of data from fine-grained monitoring and profiling tools [4,5]. Related techniques have been proposed to help detect a performance problem and identify symptoms associated with the problem [21, 25, 56, 72, 78]. For example, Collectl [4] provides the ability to monitor a broad set of system level metrics such as CPU and I/O operations at millisecond-level granularity. Chopstix [21] continuously collects profiles of low-level OS events (e.g., scheduling, L2 cache misses, page allocation, locking) at the granularity of executables, procedures and instruction. Li et al. [56] propose a fine-grained timestamping technique to measure how much time a request spends in different parts of the server OS. We use these tools when applicable.

Techniques based on end-to-end request-flow tracing have been proposed for performance anomaly diagnosis [8, 15, 26, 36, 77, 80], but usually for more stable and longer phenomena. X-ray [15] instruments binaries as applications execute and uses dynamic information flow tracking to estimate the likelihood that a block was executed due to each potential root cause for the performance anomaly. Fay [36] provides dynamic tracing through use of run-time instrumentation and distributed aggregation within machines and across clusters for windows platform. Aguilera et al. [8] infer causal paths between component servers in a distributed system and attribute delays to specific nodes. Pip [77] detects anomalous requests by comparing request-flows from actual behaviors with developer-expected behaviors. Spectroscope [80] is similar to Pip, but Spectroscope compares request-flows between "problem" periods and "non-problem" periods for identifying anomalous requests.

Analytical models have been proposed for performance analysis and prediction of n-tier

systems. Magpie [18] extracts the component control flow and resource consumption of each request to build a workload model for performance prediction. Urgaonkar [88] present a flexible queuing model for an n-tier application that determines how much resources to allocate to each tier of the application for the target system response time; Cohen [30] use a class of probabilistic models to correlate system-level metrics and threshold values with high-level performance states. Though they have been shown to be accurate when the system resource utilization is low, they may fail when the system has latency long-tail problem caused by frequent transient bottlenecks.

CHAPTER VI

CONCLUSION AND FUTURE WORK

An increasing number of enterprises are moving their applications to cloud platforms. This rapid growth in computing clouds and datacenters has accelerated the adoption of virtualization technologies, which aims to reduce the cost of operation and maximize profit by enabling horizontal and vertical scaling within a single physical host. Through hardware virtualization, consolidated servers each with specific number of core allotment run on the same physical node in dedicated Virtual Machines (VMs) to increase overall node utilization which increases profit by reducing operational costs [38]. Unfortunately, despite the conceptual simplicity of vertical and horizontal scaling in virtualized cloud environments, leveraging the full potential of this technology has presented significant scalability challenges in practice. In fact, enterprise computing infrastructures continue to struggle with surprisingly low resource utilization [31, 84]. One of the fundamental problem is the performance unpredictability in virtualized cloud environments (ranked fifth in the top 10 obstacles for growth of cloud computing [14]).

There are numerous factors that may contribute to the apparent unpredictability of n-tier application performance (e.g., response time, throughput) when scaling in virtualized cloud environments. These factors, which we call Quality of Services (QoS) determinants, include but are not limited to hardware and software components (e.g., multi-core processor, hypervisor, server thread pool size), system state (e.g., data size and distribution),and workload (e.g., workload transaction mix). For example, CPU cache miss (e.g., Last Level Cache miss) or application level cache miss (e.g., database cache or web server cache) create significant variance in performance measurement making it difficult to provide predictable application performance [54, 69, 76]; Performance interference among consolidated applications have been demonstrated variety of concrete systems and applications [11, 42, 51, 73]

adding more difficulties. Due to the complex dependencies among the hardware and software components in the system [93], the unpredictability of n-tier application performance when scaling in virtualized cloud environments is a compound effect of all these factors.

In my dissertation research, we present two case studies in vertical and horizontal scaling to this challenging problem. For the first case study, we describe concrete experimental evidence that shows important source of performance variations: mapping of virtual [1] CPU (vCPU) to physical cores. We identified mapping of vCPUs to Physical cores is an important source of performance variation within MCP due to its significant influence on cache hit ratio (see Section 2.3). After we eliminate these variations by fixing the MCP core mapping, we investigated the impact of three mainstream hypervisors (i.e., the dominant Commercial Hypervisor [2] (CH), KVM [1], and Xen [2]) with regard to their support of n-tier applications running on multi-core processor. Our results presented interesting similarities and dissimilarities among the three hypervisors. For similarity, we found a non-monotonic scalability trend (throughput increasing up to 4 cores) when running a browse-only CPU intensive workload. This problem can be traced to the hypervisors' inefficient management of last level cache (LLC) of CPU packages(see Section 2.4). For dissimilarities, we found that each hypervisor's startegy of handling write operations can cause significant performance differences when running a mixed read/write I/O-intensive workloads (see Section 2.5).

For the second case study, we investigated the performance interference among consolidated VMs through horizontal scalability of RUBBoS n-tier benchmark. We identified that the memory thrashing caused by virtual machine consolidation is an important source of unpredictable performance due to its significant influence on hypervisors' efficiency in IO request management. For example, we found system was only able to scale linearly up to three database virtual machines and with four database virtual machines, the throughput deteriorates nearly 50% compared to that of three database virtual machines despite the moderate system utilization. On a physical host with four database server virtual machines

[1] We use the term *virtual* to refer to hardware components of a virtual machine (VM).

[2] Due to licensing and copyrights issues which prevent publications of performance or comparison data, we mask our choice of commercial virtualization technology. We use commercial hypervisor or CH interchangeably throughout the proposal.

consolidated, we observed two distinct operational modes during a typical RUBBoS benchmark experiment. Over the first half of run-time session, we found frequent CPU IOwait causing very long response time requests even though the system is under browse-only CPU intensive workload; however, the latter half showed no such CPU abnomalities (IOwait). This problem can be traced to the hypervisor's inefficient management of IO requests when there is a surge induced by a large number of page faults (see Section 3.3)

We then present three practical techniques that reduce its severity or mitigate its effect on the whole system performance. For example, we show that 1) VM migration and 2) memory reallocation resolves the performance interference when more computing power is available. We also show that by reducing the concurrency of the system through adjusting 3) soft resource allocation, we can remedy the interference when more resources are unavailable. (see Section 3.4).

Lastly we conducted experimental study of very short bottleneck induced by memory thrashing. Applying a transient event analysis on extensive experimental data collected from fine-grain monitoring of n-tier application benchmarks, we demonstrate that the very long response time (VLRT) requests may arise from memory thrashing among consolidated virtual machines (VMs). These phenomena can be modeled and described as *very short bottlenecks*, very short periods of time (tens to hundreds of milliseconds) in which the CPU is saturated. The transient event analysis shows that the very long response time requests are coincidental to very short bottlenecks in database servers caused by memory thrashing, which in turn propagate and amplify queuing in upstream servers, quickly leading to TCP buffer overflow and frequent request re-transmission, resulting very long response time requests of several seconds (see Section 4.3).

By combining fine-grained monitoring tools (a combination of microsecond resolution message time-stamping and millisecond system resource sampling) and empirical methods to generate and analyze monitoring data, we are able to study, understand, and address scalability challenges in a systematic way.

6.1 Future Work

Due to the potential depth of the proposed research, the proposed dissertation can merely be regarded as an initial step towards one of the important goals in virtualized cloud environments: achieving good performance through combination of vertical and horizontal scaling strategy. One topic of particular interest is how different underlying virtualization technologies support horizontal scaling of n-tier application. Moreover how can we decide a good scaling strategy on cloud environments? Specifically, which combination of server consolidation strategy, core allocation strategy, and hypervisor choice achieves the most performance with minimal cost? I expect further research could lead us to the holy grail of performance scalability of n-tier application in virtualized cloud environments.

REFERENCES

[1] *Kernel Based Virtual Machine*. "http://www.linux-kvm.org/".

[2] *Xen*. "http://www.xenproject.org/".

[3] *Rice University Bulletin Board System*. "http://jmob.ow2.org/rubbos.html", 2004.

[4] *collectl: Versatile Resource monitoring Tool*. "http://collectl.sourceforge.net/", 2010.

[5] "Oprofile." "http://oprofile.sourceforge.net/".

[6] Aberer, K., Risse, T., and Wombacher, A., "Configuration of distributed message converter systems using performance modeling," in *Proceedings of 20th International Performance, Computation and Communication Conference (IPCCC2001), IEEE Computer Society*, Citeseer, 2001.

[7] Adams, K. and Agesen, O., "A comparison of software and hardware techniques for x86 virtualization," in *Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XII, (New York, NY, USA), pp. 2–13, ACM, 2006.

[8] Aguilera, M. K., Mogul, J. C., Wiener, J. L., Reynolds, P., and Muthitacharoen, A., "Performance debugging for distributed systems of black boxes," in *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, SOSP '03, (New York, NY, USA), pp. 74–89, ACM, 2003.

[9] Alizadeh, M., Greenberg, A., Maltz, D. A., Padhye, J., Patel, P., Prabhakar, B., Sengupta, S., and Sridharan, M., "Data center TCP (DCTCP)," in *Proceedings of the ACM SIGCOMM 2010 Conference*, pp. 63–74, 2010.

[10] Alizadeh, M., Kabbani, A., Edsall, T., Prabhakar, B., Vahdat, A., and Yasuda, M., "Less is more: Trading a little bandwidth for ultra-low latency in the data center," in *Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI'12)*, pp. 253–266, 2012.

[11] Apparao, P., Iyer, R., Zhang, X., Newell, D., and Adelmeyer, T., "Characterization & analysis of a server consolidation benchmark," in *Proceedings of the Fourth ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, VEE '08, (New York, NY, USA), pp. 21–30, ACM, 2008.

[12] Apparao, P., Iyer, R., Zhang, X., Newell, D., and Adelmeyer, T., "Characterization & analysis of a server consolidation benchmark," in *Proceedings of the Fourth ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, VEE '08, (New York, NY, USA), pp. 21–30, ACM, 2008.

[13] Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., and others, "A view of cloud computing," *Communications of the ACM*, vol. 53, no. 4, pp. 50–58, 2010.

[14] Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., and Zaharia, M., "A view of cloud computing," *Commun. ACM*, vol. 53, pp. 50–58, Apr. 2010.

[15] Attariyan, M., Chow, M., and Flinn, J., "X-ray: Automating root-cause diagnosis of performance anomalies in production software," in *Proceedings of the 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI '12)*, pp. 307–320, 2012.

[16] Attariyan, M. and Flinn, J., "Automating configuration troubleshooting with dynamic information flow analysis," in *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*, OSDI'10, (Berkeley, CA, USA), pp. 237–250, USENIX Association, 2010.

[17] Banerjee, I., Guo, F., Tati, K., and Venkatasubramanian, R., "Memory overcommitment in the esx server," *VMware Technical Journal*, vol. 2, pp. 2–12, June 2013.

[18] Barham, P., Donnelly, A., Isaacs, R., and Mortier, R., "Using magpie for request extraction and workload modelling," in *OSDI'04*, USENIX Association, 2004.

[19] Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., and Warfield, A., "Xen and the art of virtualization," in *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP 2003)*, pp. 164–177, 2003.

[20] Beltran, V., Torres, J., and Ayguade, E., "Understanding tuning complexity in multithreaded and hybrid web servers," in *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*, pp. 1–12, April 2008.

[21] Bhatia, S., Kumar, A., Fiuczynski, M. E., and Peterson, L., "Lightweight, high-resolution monitoring for troubleshooting production systems," in *Proceedings of the 8th USENIX Symposium on Operating Systems Design and Implementation (OSDI '08)*, pp. 103–116, 2008.

[22] Bodik, P., Fox, A., Franklin, M. J., Jordan, M. I., and Patterson, D. A., "Characterizing, modeling, and generating workload spikes for stateful services," in *Proceedings of the 1st ACM Symposium on Cloud Computing*, SoCC '10, (New York, NY, USA), pp. 241–252, ACM, 2010.

[23] Boyd-Wickizer, S., Clements, A. T., Mao, Y., Pesterev, A., Kaashoek, M. F., Morris, R., and Zeldovich, N., "An analysis of linux scalability to many cores," in *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*, OSDI'10, (Berkeley, CA, USA), pp. 1–16, USENIX Association, 2010.

[24] Camargos, F. L., Girard, G., and Ligneris, B. D., "Virtualization of linux servers: a comparative study," in *Proceedings of the Linux Symposium*, pp. 63–76, 2008.

[25] Cantrill, B., Shapiro, M. W., and Leventhal, A. H., "Dynamic instrumentation of production systems.," in *Proceedings of the 2004 USENIX Annual Technical Conference*, pp. 15–28, 2004.

[26] Chen, M. Y., Kiciman, E., Fratkin, E., Fox, A., and Brewer, E., "Pinpoint: Problem determination in large, dynamic internet services," in *Proceedings of the 32th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2002)*, pp. 595–604, 2002.

[27] Cho, S. and Jin, L., "Managing distributed, shared l2 caches through os-level page allocation," in *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 39, (Washington, DC, USA), pp. 455–468, IEEE Computer Society, 2006.

[28] Choi, K., Soma, R., and Pedram, M., "Fine-grained dynamic voltage and frequency scaling for precise energy and performance trade-off based on the ratio of off-chip access to on-chip computation times," in *Proceedings of the 2004 Design, Automation and Test in Europe Conference and Exposition (DATE 2004)*, pp. 4–9, 2004.

[29] Clark, B., Deshane, T., Dow, E., Evanchik, S., Finlayson, M., Herne, J., and Matthews, J. N., "Xen and the art of repeated research," in *Proceedings of the Annual Conference on USENIX Annual Technical Conference*, ATEC '04, (Berkeley, CA, USA), pp. 47–47, USENIX Association, 2004.

[30] Cohen, I., Chase, J. S., Goldszmidt, M., Kelly, T., and Symons, J., "Correlating instrumentation data to system states: A building block for automated diagnosis and control.," in *Proceedings of the 6th USENIX Symposium on Operating Systems Design and Implementation (OSDI '04)*, pp. 231–244, 2004.

[31] Curino, C., Jones, E. P., Madden, S., and Balakrishnan, H., "Workload-aware database monitoring and consolidation," in *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*, SIGMOD '11, (New York, NY, USA), pp. 313–324, ACM, 2011.

[32] Dean, J. and Barroso, L. A., "The tail at scale," *Commun. ACM*, vol. 56, pp. 74–80, Feb. 2013.

[33] Dean, J. and Barroso, L. A., "The tail at scale," *Commun. ACM*, vol. 56, pp. 74–80, Feb. 2013.

[34] Deshane, T., Shepherd, Z., Matthews, J., Ben-Yehuda, M., Shah, A., and Rao, B., "Quantitative comparison of Xen and KVM," in *Xen summit*, (Berkeley, CA, USA), USENIX association, June 2008.

[35] Diao, Y., Hellerstein, J., Storm, A., Surendra, M., Lightstone, S., Parekh, S., and Garcia-Arellano, C., "Using MIMO linear control for load balancing in computing systems," in *American Control Conference*, pp. 2045–2050, 2004.

[36] Erlingsson, Ú., Peinado, M., Peter, S., Budiu, M., and Mainar-Ruiz, G., "Fay: Extensible distributed tracing from kernels to clusters," *ACM Transactions on Computer Systems (TOCS)*, vol. 30, no. 4, p. 13, 2012.

[37] Ferreto, T. C., Netto, M. A. S., Calheiros, R. N., and De Rose, C. A. F., "Server consolidation with migration control for virtualized data centers," *Future Gener. Comput. Syst.*, vol. 27, pp. 1027–1034, Oct. 2011.

[38] Forell, T., Milojicic, D., and Talwar, V., "Cloud management: Challenges and opportunities," in *Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW), 2011 IEEE International Symposium on*, pp. 881–889, May 2011.

[39] Franks, G., Petriu, D., Woodside, M., Xu, J., and Tregunno, P., "Layered bottlenecks and their mitigation," in *QEST '06: Proceedings of the 3rd international conference on the Quantitative Evaluation of Systems*, 2006.

[40] Freeh, V. W., Lowenthal, D. K., Pan, F., Kappiah, N., Springer, R., Rountree, B. L., and Femal, M. E., "Analyzing the energy-time trade-off in high-performance computing applications," *IEEE Transactions on Parallel and Distributed Systems*, vol. 18, no. 6, pp. 835–848, 2007.

[41] Gong, Z. and Gu, X., "Pac: Pattern-driven application consolidation for efficient cloud computing," in *Proceedings of the 2010 IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, MASCOTS '10, (Washington, DC, USA), pp. 24–33, IEEE Computer Society, 2010.

[42] Gupta, D., Cherkasova, L., Gardner, R., and Vahdat, A., "Enforcing performance isolation across virtual machines in xen," in *Proceedings of the 7th ACM/IFIP/USENIX International Conference on Middleware*, Middleware'06, (Berlin, Heidelberg), pp. 342–362, Springer-Verlag, 2006.

[43] Gupta, D., Cherkasova, L., Gardner, R., and Vahdat, A., "Enforcing performance isolation across virtual machines in xen," in *Proceedings of the 7th ACM/IFIP/USENIX International Conference on Middleware*, Middleware'06, (Berlin, Heidelberg), pp. 342–362, Springer-Verlag, 2006.

[44] Gupta, D., Lee, S., Vrable, M., Savage, S., Snoeren, A. C., Varghese, G., Voelker, G. M., and Vahdat, A., "Difference engine: Harnessing memory redundancy in virtual machines," in *Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation*, OSDI'08, (Berkeley, CA, USA), pp. 309–322, USENIX Association, 2008.

[45] Isci, C., Contreras, G., and Martonosi, M., "Live, runtime phase monitoring and prediction on real systems with application to dynamic power management," in *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-39)*, pp. 359–370, 2006.

[46] Jain, R., *The art of computer systems performance analysis - techniques for experimental design, measurement, simulation, and modeling.* Wiley professional computing, Wiley, 1991.

[47] Jayasinghe, D., Malkowski, S., Wang, Q., Li, J., Xiong, P., and Pu, C., "Variations in performance and scalability when migrating n-tier applications to different clouds," in *Cloud Computing (CLOUD), 2011 IEEE International Conference on*, pp. 73–80, July 2011.

[48] Kanemasa, Y., Wang, Q., Li, J., Matsubara, M., and Pu, C., "Revisiting performance interference among consolidated n-tier applications: Sharing is better than isolation," in *Services Computing (SCC), 2013 IEEE International Conference on*, pp. 136–143, June 2013.

[49] Kapoor, R., Porter, G., Tewari, M., Voelker, G. M., and Vahdat, A., "Chronos: Predictable low latency for data center applications," in *Proceedings of the Third ACM Symposium on Cloud Computing*, SoCC '12, (New York, NY, USA), pp. 9:1–9:14, ACM, 2012.

[50] Kapoor, R., Porter, G., Tewari, M., Voelker, G. M., and Vahdat, A., "Chronos: Predictable low latency for data center applications," in *Proceedings of the 3rd ACM Symposium on Cloud Computing (SoCC 2012)*, pp. 9:1–9:14, 2012.

[51] Koh, Y., Knauerhase, R., Brett, P., Bowman, M., Wen, Z., and Pu, C., "An analysis of performance interference effects in virtual environments," in *Performance Analysis of Systems Software, 2007. ISPASS 2007. IEEE International Symposium on*, pp. 200–209, April 2007.

[52] Koh, Y., Knauerhase, R., Brett, P., Bowman, M., Wen, Z., and Pu, C., "An analysis of performance interference effects in virtual environments," in *Performance Analysis of Systems Software, 2007. ISPASS 2007. IEEE International Symposium on*, pp. 200–209, April 2007.

[53] Lai, C. A., Wang, Q., Kimball, J., Li, J., Park, J., and Pu, C., "Io performance interference among consolidated n-tier applications: Sharing is better than isolation for disks," in *Cloud Computing (CLOUD), 2014 IEEE 7th International Conference on*, pp. 24–31, June 2014.

[54] Lee, M. and Schwan, K., "Region scheduling: Efficiently using the cache architectures via page-level affinity," in *Proceedings of the Seventeenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XVII, (New York, NY, USA), pp. 451–462, ACM, 2012.

[55] Leverich, J. and Kozyrakis, C., "Reconciling high server utilization and sub-millisecond quality-of-service," in *Proceedings of the Ninth European Conference on Computer Systems*, EuroSys '14, (New York, NY, USA), pp. 4:1–4:14, ACM, 2014.

[56] Li, J., Sharma, N. K., Ports, D. R., and Gribble, S. D., "Tales of the tail: Hardware, os, and application-level sources of tail latency," Tech. Rep. UW-CSE14-04-01, Department of Computer Science & Engineering, University of Washington, April 2014.

[57] Lilja, D. J., *Measuring computer performance : a practitioner's guide*. Cambridge, New York, Merlbourne: Cambridge University Press, 2000. Appendices. Index.

[58] Liu, X., Sha, L., Diao, Y., Froehlich, S., Hellerstein, J. L., and Parekh, S., "Online response time optimization of apache web server," pp. 461–478, 2003.

[59] Malkowski, S., Hedwig, M., and Pu, C., "Experimental evaluation of n-tier systems: Observation and analysis of multi-bottlenecks," in *Workload Characterization, 2009. IISWC 2009. IEEE International Symposium on*, pp. 118–127, Oct 2009.

[60] Malkowski, S., Kanemasa, Y., Chen, H., Yamamoto, M., Wang, Q., Jayasinghe, D., Pu, C., and Kawaba, M., "Challenges and opportunities in consolidation at high resource utilization: Non-monotonic response time variations in n-tier applications," in *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, pp. 162–169, June 2012.

[61] Meng, X., Isci, C., Kephart, J., Zhang, L., Bouillet, E., and Pendarakis, D., "Efficient resource provisioning in compute clouds via vm multiplexing," in *Proceedings of the 7th International Conference on Autonomic Computing*, ICAC '10, (New York, NY, USA), pp. 11–20, ACM, 2010.

[62] Merkel, A., Stoess, J., and Bellosa, F., "Resource-conscious scheduling for energy efficiency on multicore processors," in *Proceedings of the 5th European conference on Computer systems (EuroSys 2010)*, pp. 153–166, 2010.

[63] Mi, N., Casale, G., Cherkasova, L., and Smirni, E., "Injecting realistic burstiness to a traditional client-server benchmark," in *Proceedings of the 6th International Conference on Autonomic computing (ICAC 2009)*, pp. 149–158, 2009.

[64] Miftakhutdinov, R., Ebrahimi, E., and Patt, Y. N., "Predicting Performance Impact of DVFS for Realistic Memory Systems," in *Proceedings of the 45th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-45)*, 2012.

[65] Oliveira, F., Tjang, A., Bianchini, R., Martin, R. P., and Nguyen, T. D., "Barricade: Defending systems against operator mistakes," in *Proceedings of the 5th European Conference on Computer Systems*, EuroSys '10, (New York, NY, USA), pp. 83–96, ACM, 2010.

[66] Olshefski, D. and Nieh, J., "Understanding the management of client perceived response time," in *Proceedings of the Joint International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '06/Performance '06, (New York, NY, USA), pp. 240–251, ACM, 2006.

[67] Osogami, T. and Kato, S., "Optimizing system configurations quickly by guessing at the performance," in *Proceedings of the 2007 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '07, (New York, NY, USA), pp. 145–156, ACM, 2007.

[68] Pariag, D., Brecht, T., Harji, A., Buhr, P., Shukla, A., and Cheriton, D. R., "Comparing the performance of web server architectures," *In Proc. EuroSys '07*, 2007.

[69] Park, J., Wang, Q., Jayasinghe, D., Li, J., Kanemasa, Y., Matsubara, M., Yokoyama, D., Kitsuregawa, M., and Pu, C., "Variations in performance measurements of multi-core processors: A study of n-tier applications," in *Services Computing (SCC), 2013 IEEE International Conference on*, pp. 336–343, June 2013.

[70] Patterson, D. A., "Latency lags bandwith," *Commun. ACM*, vol. 47, pp. 71–75, Oct. 2004.

[71] Paul, I., Yalamanchili, S., and John, L., "Performance impact of virtual machine placement in a datacenter," in *Performance Computing and Communications Conference (IPCCC), 2012 IEEE 31st International*, pp. 424–431, Dec 2012.

[72] Prasad, V., Cohen, W., Eigler, F., Hunt, M., Keniston, J., and Chen, B., "Locating system problems using dynamic instrumentation," in *Proceedings of the 2005 Ottawa Linux Symposium*, pp. 49–64, 2005.

[73] Pu, X., Liu, L., Mei, Y., Sivathanu, S., Koh, Y., and Pu, C., "Understanding performance interference of i/o workload in virtualized cloud environments," in *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*, pp. 51–58, July 2010.

[74] Pu, X., Liu, L., Mei, Y., Sivathanu, S., Koh, Y., and Pu, C., "Understanding performance interference of i/o workload in virtualized cloud environments," in *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*, pp. 51–58, July 2010.

[75] Raghavachari, M., Reimer, D., and Johnson, R. D., "The deployer's problem: Configuring application servers for performance and reliability," pp. 484–489, 2003.

[76] Rasmussen, A., Kiciman, E., Livshits, B., and Musuvathi, M., "Improving the responsiveness of internet services with automatic cache placement," in *Proceedings of the 4th ACM European Conference on Computer Systems*, EuroSys '09, (New York, NY, USA), pp. 27–32, ACM, 2009.

[77] Reynolds, P., Killian, C. E., Wiener, J. L., Mogul, J. C., Shah, M. A., and Vahdat, A., "Pip: Detecting the unexpected in distributed systems.," in *Proceedings of the 3rd USENIX Symposium on Networked Systems Design and Implementation (NSDI'06)*, pp. 115–128, 2006.

[78] Ruan, Y. and Pai, V. S., "Making the" box" transparent: System call performance as a first-class result.," in *Proceedings of the 2004 USENIX Annual Technical Conference*, pp. 1–14, 2004.

[79] Rumble, S. M., Ongaro, D., Stutsman, R., Rosenblum, M., and Ousterhout, J. K., "It's time for low latency," in *Proceedings of the 13th USENIX Workshop on Hot Topics in Operating Systems (HotOS 13)*, pp. 11–11, 2011.

[80] Sambasivan, R. R., Zheng, A. X., De Rosa, M., Krevat, E., Whitman, S., Stroucken, M., Wang, W., Xu, L., and Ganger, G. R., "Diagnosing performance changes by comparing request flows," in *Proceedings of the 8th USENIX Symposium on Networked Systems Design and Implementation (NSDI'11)*, pp. 43–56, 2011.

[81] Sen, A., Garg, A., Verma, A., and Nayak, T., "Cloudbridge: On integrated hardware-software consolidation," *SIGMETRICS Perform. Eval. Rev.*, vol. 39, pp. 14–25, Sept. 2011.

[82] Sharma, U., Shenoy, P., and Towsley, D. F., "Provisioning multi-tier cloud applications using statistical bounds on sojourn time," in *Proceedings of the 9th International Conference on Autonomic Computing*, ICAC '12, (New York, NY, USA), pp. 43–52, ACM, 2012.

[83] Snowdon, D. C., Le Sueur, E., Petters, S. M., and Heiser, G., "Koala: A platform for OS-level power management," in *Proceedings of the 4th ACM European conference on Computer systems (EuroSys 2009)*, pp. 289–302, 2009.

[84] Snyder, B., "Server virtualization has stalled, despite the hype," *InfoWorld*, 2010.

[85] Song, X., Chen, H., and Zang, B., "Characterizing the performance and scalability of many-core applications on virtualized platforms," in *Proceedings of Parallel Processing Insitute*, PPITR-2011-001.

[86] Stewart, C. and Shen, K., "Performance modeling and system management for multi-component online services," in *Proceedings of the 2Nd Conference on Symposium on Networked Systems Design & Implementation - Volume 2*, NSDI'05, (Berkeley, CA, USA), pp. 71–84, USENIX Association, 2005.

[87] Urgaonkar, B., Pacifici, G., Shenoy, P., Spreitzer, M., and Tantawi, A., "An analytical model for multi-tier internet services and its applications," in *Proceedings of the 2005 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '05, (New York, NY, USA), pp. 291–302, ACM, 2005.

[88] Urgaonkar, B., Shenoy, P., Chandra, A., and Goyal, P., "Dynamic provisioning of multi-tier internet applications," ICAC '05.

[89] Veal, B. and Foong, A., "Performance scalability of a multi-core web server," in *Proceedings of the 3rd ACM/IEEE Symposium on Architecture for Networking and Communications Systems*, ANCS '07, (New York, NY, USA), pp. 57–66, ACM, 2007.

[90] Wang, Q., Kanemasa, Y., Li, J., Jayasinghe, D., Shimizu, T., Matsubara, M., Kawaba, M., and Pu, C., "Detecting transient bottlenecks in n-tier applications through fine-grained analysis," in *Distributed Computing Systems (ICDCS), 2013 IEEE 33rd International Conference on*, pp. 31–40, July 2013.

[91] Wang, Q., Kanemasa, Y., Kawaba, M., and Pu, C., "When average is not average: Large response time fluctuations in n-tier systems," in *Proceedings of the 9th International Conference on Autonomic Computing*, ICAC '12, (New York, NY, USA), pp. 33–42, ACM, 2012.

[92] Wang, Q., Kanemasa, Y., Li, J., Lai, C.-A., Cho, C.-A., Nomura, Y., and Pu, C., "Lightning in the cloud: A study of transient bottlenecks on n-tier web application performance," in *2014 Conference on Timely Results in Operating Systems (TRIOS 14)*, (Broomfield, CO), USENIX Association, Oct. 2014.

[93] Wang, Q., Malkowski, S., Kanemasa, Y., Jayasinghe, D., Xiong, P., Pu, C., Kawaba, M., and Harada, L., "The impact of soft resource allocation on n-tier application scalability," in *Parallel Distributed Processing Symposium (IPDPS), 2011 IEEE International*, pp. 1034–1045, May 2011.

[94] Wang, Q., Malkowski, S., Kanemasa, Y., Jayasinghe, D., Xiong, P., Pu, C., Kawaba, M., and Harada, L., "The impact of soft resource allocation on n-tier application scalability," in *Parallel Distributed Processing Symposium (IPDPS), 2011 IEEE International*, pp. 1034–1045, May 2011.

[95] Welsh, M., Culler, D., and Brewer, E., "SEDA: An architecture for well-conditioned, scalable internet services," in *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP 2001)*, pp. 230–243, 2001.

[96] Xiong, K. and Perros, H., "Service performance and analysis in cloud computing," in *Services - I, 2009 World Conference on*, pp. 693–700, July 2009.

[97] Xu, Y., Musgrave, Z., Noble, B., and Bailey, M., "Bobtail: Avoiding long tails in the cloud," in *Proceedings of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI'13)*, pp. 329–342, 2013.

[98] Ye, K., Jiang, X., Ye, D., and Huang, D., "Two optimization mechanisms to improve the isolation property of server consolidation in virtualized multi-core server," in *High Performance Computing and Communications (HPCC), 2010 12th IEEE International Conference on*, pp. 281–288, Sept 2010.

[99] Zheng, W., Bianchini, R., and Nguyen, T. D., "Automatic configuration of internet services," vol. 41, (New York, NY, USA), pp. 219–229, ACM, Mar. 2007.